MASTER EN SCIENCES INFORMATIQUES

Conception d'un éditeur graphique pour le langage de spécifications D.S.L.

Crespin, Sophie; Piquard, Patrick

Award date:
1987

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix

NAMUR

Institut d'informatique

Conception d'un éditeur graphique pour le langage de spécifications D.S.L.

Sophie CRESPIN Patrick PIQUARD

Mémoire présenté en vue de l'obtention du titre de
Licencié et Maître en Informatique
1986-1987

Remerciements

Nous tenons à remercier très sincèrement nos parents pour le temps et l'attention qu'ils nous ont consacrés ainsi que la tendre affection qu'ils nous ont témoignée tout au long de nos études.

Nous remercions vivement notre promoteur, Monsieur François Bodart pour les judicieuses remarques qu'il nous a apportées lors de la conception et de la rédaction de ce mémoire.

Our truly thanks go to John Underhill, who has supported us during our stay at the University of Michigan and was always present to help us when needed. We would like to thank Doctor Teichroew, manager of the PRISE project, for providing us four marvelous months in the United States.

Nous remercions également les membres de l'équipe IDA pour leurs observations pertinentes lors de la conception, et tout particulièrement Marcel Clantin pour son aide et son continuel soutien tout au long de cette année.

Enfin, nous n'oublierons pas de remercier tous nos amis pour leur support moral qui nous a réconforté lors des moments les plus difficiles.

Table des matières

Introduction

Une part importante - environ 90 % - du temps, du travail et donc du coût nécessaires à l'entretien d'applications informatiques est consacrée à la résolution de problèmes découlant d'erreurs dans les spécifications initiales du système. Une étude faite par le Massachusetts Institute of Technology démontre que pour chaque dollar investi dans un nouveau projet de développement, jusqu'à 9 dollars doivent être dépensés pour l'entretien du système pendant son cycle de vie. Il est clair que tout ce qui peut diminuer les temps de développement et permettre aux analystes de satisfaire aux besoins des utilisateurs doit être accueilli à bras ouverts par le personnel informatique et les utilisateurs. L'emploi d'un éditeur graphique pour l'analyse conceptuelle élimine pour les analystes un ensemble de tâches fastidieuses : élaboration de diagrammes, traduction de leur analyse dans un langage de spécifications conceptuelles, ... Ils pourront donc ainsi mieux concentrer leur attention sur les besoins de l'entreprise.

L'évolution rapide des moyens informatiques tels que les possibilités de graphiques haute résolution, les ordinateurs personnels multi-tâches de plus en plus performants, la capacité des mémoires centrales et auxiliaires sans cesse croissante, ... a permis le développement d'applications graphiques élaborées. Grâce à cette évolution, ces applications peuvent être utilisées fréquemment puisque leurs temps d'exécution et de réponse sont courts.

L'analyste dispose de plusieurs modèles pour définir un système d'information. La plupart de ces modèles peuvent se représenter sous forme d'un graphique, ce qui donne une vue plus globale du travail effectué. Le logiciel IDA (Interactive Design Approach) développé aux Facultés Universitaires de Namur sous la direction du Professeur François Bodart, ne comporte pas actuellement d'éditeur graphique. L'analyste doit donc encoder ses spécifications via l'éditeur texte, et ce dans le langage de spécification D.S.L. (Dynamic Specification Language). Malgré les avantages de cet éditeur, il serait très intéressant de le compléter par un éditeur qui permettrait l'encodage de graphiques.

Pour développer un tel système, il faut tout d'abord analyser les différentes possibilités qu'offre le traitement graphique à l'utilisateur et les contraintes que le concepteur doit respecter pour obtenir une interface homme-machine la plus ergonomique possible. Différents aspects du traitement graphique qui ont retenu notre attention sont exposés à titre introductifs dans le premier chapitre.

Ensuite, nous nous sommes interrogés au sujet de la méthode de travail de l'analyste afin de comprendre les améliorations qu'un éditeur graphique pourraient lui apporter. Nous n'avancerons ici que des hypothèses que nous n'avons malheureusement pas pu vérifier. Leur vérification constitue un travail en soi.

Les chapitres suivants représentent l'essentiel de notre travail et sont consacrés à l'examen et au développement d'éléments d'un éditeur graphique.

Dans le troisième chapitre, nous analysons deux éditeurs de saisie graphique qui sont commercialisés. Nous en dégageons leurs principaux défauts et qualités. Nous exposons ensuite la structure et l'architecture de ce que nous appellerons notre "Contribution au développement d'un éditeur graphique dédié à IDA". Cette approche a été développée aux Facultés avec l'aide de l'équipe IDA. Nous terminons ce chapitre en présentant un éditeur beaucoup plus ouvert, le SEEOF (Software Engineering

Environment Of the Future), développé à l'Université du Michigan par le projet de recherche PRISE, dirigé par le Docteur Teichroew. Nous avons eu la chance de travailler quatre mois à ce projet lors de notre stage aux Etats-Unis, où nous avons développé la partie Communication Homme-Machine du SEEOF, à savoir le DES (Display Edit Server).

Dans le quatrième chapitre, nous exposons une analyse de deux systèmes de restitution existants pour en découvrir leurs avantages et inconvénients. Ceci nous amènera à définir l'architecture du système de restitution auquel nous avons collaboré. Nous expliquons le chemin que nous avons suivi pour finalement arriver au système que nous avons développé. Ici, nous ne nous sommes préoccupés que du modèle E.R.A. (Entities-Relationships-Attributes).

Nous présentons dans le cinquième chapitre les spécifications externes et internes des modules les plus importants. Le lecteur intéressé consultera dans le tome annexe, les spécifications complètes de chaque fonction.

En annexe de ce premier volume, nous présentons les grandes fonctionnalités du "Development Toolkit" de MS-WINDOWS. Nous nous sommes placés du côté du concepteur pour décrire les principales fonctions que nous avons utilisées au cours de l'élaboration de deux applications.

Chapitre 1. Le traitement graphique

1.1. Introduction

Dans ce chapitre, nous présenterons tout d'abord brièvement les différents standards graphiques qui existent ou qui sont en cours de standardisation [MEAD86] [WRIG86].

Nous aborderons ensuite les aspects généraux du traitement graphique, ses fonctionnalités et ses apports dans la communication homme-machine [SING83] [RAED85]. Nous n'avons pas la prétention d'être exhaustifs en ce domaine, notre seul but étant de sensibiliser le lecteur aux aspects qui déterminent le contexte dans lequel nous travaillerons.

Les logiciels graphiques permettent l'utilisation de moyens de traitement beaucoup plus faciles et rapides à apprendre tels que les menus qui donnent les options du système sans que l'utilisateur ait à les mémoriser, un système utilisant des fenêtres qui permettent une visualisation simultanée de différentes applications, la transparence du système grâce à laquelle l'utilisateur visualise parfaitement les objets à manipuler, la facilité de transition d'une application à l'autre et enfin la possibilité de combiner du texte et des graphiques sur un même écran.

Grâce au traitement graphique, la communication homme-machine s'est fortement améliorée. L'Homme acquiert et mémorise beaucoup plus d'informations en analysant un graphique et les différentes relations existant entre les objets représentés qu'en analysant un texte. De plus, l'utilisateur peut accéder à l'information de manière aléatoire, alors que le texte ne lui permettait qu'un accès séquentiel.

Enfin, nous parlerons de l'ergonomie intellectuelle de l'utilisateur. Pour ce faire, nous analyserons sa psychologie et son comportement vis-à-vis d'un logiciel graphique. Pour plus d'informations à ce sujet, le lecteur intéressé pourra consulter [CLAN83], [MORA81] et [SING83].

1.2. Les standards graphiques

1.2.1. Les raisons des standards graphiques

Il existe principalement deux buts pour les standardisations. Le premier est de promouvoir l'interchangeabilité des composants, l'autre de fournir une mesure de qualité, qui a deux aspects : les services et la performance.

L'identification et la spécification des composants d'un système sont considérées comme un modèle de référence pour ce système. Dans un bon modèle de référence, chaque composant est distinct des autres et a un rôle et un but dans le système qui sont séparés et complémentaires vis-à-vis des autres composants. L'identification des composants et de leurs inter-relations établit les interfaces qui ont besoin d'être définies et standardisées.

1.2.2. Les principaux standards

* **GKS : Graphical Kernel System.**

GKS est un ensemble de fonctions de base pour la programmation graphique. C'est un standard établi et il est habituellement utilisé pour des applications qui ont besoin de générer des diagrammes de bonne qualité. Cependant, GKS est limité. Il ne pourvoit pas un support adéquat pour les graphiques dynamiques et ne supporte pas les conversations multiples d'un utilisateur avec d'autres applications indépendantes (par exemple, il ne travaille pas, comme défini, dans un environnement multi-fenêtres).

L'interface définie par GKS est de bas niveau. Sa force réside dans son ampleur technique et dans la complétude de ses utilités de support plutôt que sur ses facilités de dessin.

Pour développer une application, il n'existe que cinq fonctions graphiques et six de saisie. Mais il existe deux cents routines pour contrôler le système, les attributs des primitives graphiques, l'état du système graphique, etc ...

Le plus gros désavantage de GKS est qu'il est très compliqué à utiliser.

*** GKS-3D**

GKS-3D est une extension du GKS standard. Il permet la réalisation d'images en trois dimensions. GKS-3D est encore plus complexe à utiliser que GKS.

*** PHIGS : Programmer's Hierarchical Interactive Graphics System**

Ce standard donne une spécification fonctionnelle de l'interface entre un programme d'application et son système de support graphique. Il est spécialement orienté vers le support des objets géométriques en deux ou trois dimensions, qui peuvent être affichés dynamiquement. Sa complexité est également son plus gros défaut.

*** CGM : Computer Graphics Metafile**

CGM devrait contribuer au format de fichier compatible pour la sauvegarde et le chargement d'une information graphique. Il permettrait d'associer des images de différentes origines dans un même graphique.

*** CGI : Computer Graphics Interface**

CGI est un ensemble d'éléments de base permettant le contrôle de l'échange des données entre la couche logique et la couche physique. Au niveau physique, les données sont dépendantes des caractéristiques des périphériques telles que la résolution d'une imprimante ou d'un écran. Au niveau logique, elles en sont indépendantes.

CGI, tout comme CGM, donne un ensemble plus riche de primitives de restitution que les autres standards.

1.3. Fonctionnalités des moyens graphiques

1.3.1. La souris

Les outils de pointage (tels que la souris, la poignée de jeu, le "trackball" ou la tablette digitale) permettent à l'utilisateur de l'ordinateur de contrôler la position d'un curseur sur l'écran pour sélectionner des objets spécifiques qui ont été affichés auparavant, ou comme un crayon pour dessiner directement dans le document affiché [HAYE81].

La souris est un outil de saisie à tenir en main et qui tient dans la paume de la main. Elle comporte un élément sensoriel sur le bas pour détecter les mouvements sur une surface plane. Ces mouvements causent des changements relatifs en X et en Y qui sont prélevés par le système. Les coordonnées de la souris sont mises en correspondance avec les coordonnées de l'écran pour bouger un curseur sur celui-ci [MEYR82]. La souris est l'outil de saisie le plus utilisé. Elle est moins coûteuse que les autres outils et elle est parfaitement utilisable avec des systèmes de haute résolution graphique (il est possible de désigner un point sur un écran de 1024 sur 1024 points). De plus, une souris reste positionnée quand l'utilisateur la lâche, ce qui est une propriété importante lorsqu'il est nécessaire d'utiliser fréquemment en alternance l'outil de saisie et le clavier [WARF83].

1.3.2. Les menus

Les menus affichés à l'écran ne sont rien d'autre qu'une liste de phrases ou d'icônes. Chaque phrase affichée dans un menu représente une commande pour un système interactif, un moyen d'accéder aux fonctionnalités du système [LIEB85].

D'un menu, l'utilisateur choisit une action en bougeant la souris jusqu'à ce que le curseur soit aligné avec l'option désirée du menu sur l'écran. Appuyer sur un des boutons de la souris sélectionne l'action.

Les items affichés peuvent spécifier des actions ou peuvent faire appel à un autre menu. Un schéma basé sur les menus offre plusieurs avantages. Par exemple, cela permet à l'utilisateur de visualiser toutes les options du système qui sont disponibles. Ceci évite le problème des commandes erronées en empêchant l'utilisateur de sélectionner une option qui n'existe pas. Cela aide aussi l'utilisateur à se souvenir des options qu'il n'utilise pas fréquemment. Ce système est très flexible : un nom de menu peut être facilement changé pour s'adapter au mieux à la terminologie de l'utilisateur.

Quand ils sont combinés avec la souris comme système de saisie, les menus rendent possible une interaction homme-machine suffisamment simple pour les novices et suffisamment rapide pour les experts. L'avantage le plus bénéfique du menu par rapport aux commandes statiques est que le menu ne requiert pas une mémorisation de syntaxe de la part de l'utilisateur.

Les menus qui sont toujours affichés au même endroit sont appelés des menus statiques. L'utilisateur s'habitue à leur place d'affichage et habituellement les cherche et s'attend à les voir apparaître à cet endroit. Ceci produit une continuité visuelle et donne une idée de place dans le dialogue interactif. L'inconvénient des menus statiques est que l'utilisateur doit bouger ses yeux (et sa main pour contrôler la souris) de leur position actuelle sur l'écran vers l'endroit où le menu est affiché. Par contre, un menu dynamique apparait à la position actuelle de l'utilisateur sur l'écran, et donc évite un mouvement supplémentaire des yeux et de la main. L'hypothèse est que l'utilisateur porte son attention sur l'endroit où se trouve le curseur, sa position actuelle sur l'écran. Donc, le menu dynamique

minimise surtout le mouvement des yeux et des mains. Les menus statiques permettent à l'utilisateur d'avoir une mémoire tactile pour positionner le curseur à l'endroit adéquat, alors que les menus dynamiques minimisent la recherche visuelle.

1.3.3. Les fenêtres

Une fenêtre est une zone de l'écran dans laquelle s'exécute une application.

La haute résolution des moniteurs graphiques modernes rend possible l'affichage simultané d'une grande quantité d'information sur un même écran. Par exemple, nous pouvons montrer plusieurs activités indépendantes dans différentes fenêtres. Quand les gens travaillent, ils ont tendance à porter leur attention d'une tâche à une autre de façon tout à fait aléatoire. Le multi-fenêtrage supporte donc cette habitude.

1.3.4. La transparence du système

Quand nous utilisons un ordinateur pour manipuler des objets, nous préférons travailler aussi directement que possible sur ces objets sans que l'ordinateur ne nous gêne. Premièrement, quand une commande est initialisée, une rétroaction immédiate devrait modifier l'écran, répercutant le résultat de la commande aussitôt que celle-ci est exécutée. Deuxièmement, le principe du "WYSIWYG" ("What You See Is What You Get") spécifie que l'affichage devrait donner une image exacte et complète de l'objet ayant été manipulé [DVOR86]. Si ces critères ne sont pas respectés, nous pouvons avoir une présomption erronée et retardée de l'état du système.

1.3.5. La rétroaction

La rétroaction est un élément essentiel dans une conversation, que ce soit avec une machine ou avec une personne. Dans une conversation normale, plusieurs formes de rétroaction sont échangées automatiquement sans que les participants en prennent conscience. La rétroaction se fait par gestes, paroles, expressions du visage et par les yeux. Des blocages psychologiques pourraient se produire en l'absence de rétroaction. Les symptômes typiques sont l'ennui, la panique, la frustration et la confusion. Une rétroaction immédiate évite non seulement ces blocages psychologiques mais aussi ajoute une certaine continuité au dialogue.

Du point de vue de l'ordinateur, la rétroaction serait plutôt un historique du dialogue avec l'utilisateur. Lorsque l'utilisateur demande une action, le système lui montre constamment le chemin qu'il a dû suivre pour y arriver. Par exemple, lorsque l'utilisateur demande de sauvegarder un fichier, le nom du menu activé reste "inversé" tout le temps que dure la sauvegarde.

1.3.6. L'état du système

Un aspect très important de la représentation de l'affichage est de montrer l'état du système à tout moment. La plupart des systèmes réservent une place sur l'écran pour afficher cette information. Pourtant, cette approche comporte quelques inconvénients. Pendant le dialogue, l'attention de l'utilisateur est généralement concentré sur l'objet qu'il manipule. Tout ce qui apparait en dehors de cette zone, spécialement hors de son champ de vision, ne sera pas enregistré. Donc, le problème d'afficher l'état du système à une place fixe est que si cet endroit n'est pas proche de l'objet que l'utilisateur manipule mais dans son champ de vision, cela le distraira. La curiosité de l'utilisateur sera portée sur l'endroit où est affiché l'état du système. Après avoir lu cette information, son

attention retournera à sa position initiale. Le mouvement des yeux chaque fois que l'état du système change peut être très fatigant et très contrariant, surtout si l'état change souvent. Une meilleure façon de présenter cette information est de l'afficher à l'endroit où l'utilisateur porte son attention, et donc d'éviter le mouvement des yeux et de donner une continuité visuelle.

L'état du système considéré ici n'est pas son état interne mais le mode actuel de l'éditeur pour le modèle conceptuel de l'utilisateur. Par exemple, lorsque l'utilisateur demande de sauvegarder ou de charger un fichier, le système lui indique "Saving file ..." ou "Loading file ..." de sorte que l'utilisateur sait ce que l'ordinateur est en train d'effectuer comme opération.

1.4. La communication homme-machine

1.4.1. Le traitement graphique et le texte

Le traitement graphique est un outil puissant dans le processus de la gestion de l'information. Par exemple, un utilisateur peut parcourir une base de données graphique sans bien connaître les objets qu'il cherche. Même si l'utilisateur sait exactement ce qu'il cherche, c'est peut-être plus facile pour lui d'effectuer un "zoom" sur une base de données graphique que de spécifier un long prédicat, comme il l'aurait fait pour une base de données texte. De plus, les positions et les formes des icônes peuvent contenir de l'information difficile à sauvegarder autrement.

Les ordinateurs traditionnels montrent très peu leur état interne à l'utilisateur. Cela signifie que l'utilisateur doit se souvenir et manipuler une structure très grande, complète et abstraite à l'aide seulement de quelque allusions de l'ordinateur pour savoir si ses opérations sont correctes par rapport à l'état du système.

Les éditeurs, les tableurs et les bureaux électroniques sont des développements récents qui tentent de remédier à cette situation en utilisant des graphiques pour montrer autant que possible l'état de l'ordinateur à l'écran. Les facilités du traitement graphique haute-résolution peuvent être utilisées pour détailler l'une ou l'autre figure et la rendre plus significative, et donc augmenter le contenu informationnel de l'écran.

Un autre aspect de la technologie du traitement graphique haute-résolution est sa nature interactive. Le traitement graphique interactif est un domaine encore peu exploré, qui semble pourtant offrir des techniques sophistiquées pour une meilleure communication homme-machine.

Il est reconnu que l'esprit humain est fortement orienté vers la visualisation et que les gens acquièrent de l'information beaucoup plus rapidement en découvrant des relations graphiques dans un dessin complexe qu'en lisant un texte.

Nos yeux fournissent un accès instantané, aléatoire à n'importe quelle partie d'un graphique. Nous reconnaissons rapidement les caractéristiques que nous connaissons et il existe différentes manières pour le programmeur pour attirer notre attention vers une partie spécifique d'un dessin. Par contre, le texte est séquentiel. Quand nous examinons un texte, nous recherchons les paragraphes, les titres, les caractères écrits en gras, ... pour accélérer notre lecture. Nous agissons en fait comme en mode graphique.

Les comparaisons de dessins et de texte montrent clairement que les diagrammes transfèrent généralement l'information plus vite que le texte.

L'utilisation d'objets issus du monde réel dans un graphique qui illustrent des idées abstraites rend ces idées plus simples à imaginer et à comprendre. De bonnes images qui font des abstractions aideront non seulement les spécialistes à formuler et à communiquer leur pensée plus vite et mieux, mais aussi les novices qui tâtonnent dans un environnement non-familier.

Les objets que nous voyons, aussi bien dans le monde réel que dans les systèmes graphiques peuvent être désignés en les montrant. Leurs noms nous permettent de les désigner sans les montrer, qu'ils soient présents ou non. Les noms introduisent donc un deuxième niveau de référence qui peut être un obstacle. Les noms ne sont plus nécessaires quand nous utilisons des images,

puisque celles-ci sont présentes et qu'elles sont la représentation visible des objets.

Avec les moniteurs graphiques, la technologie semble finalement avoir éliminé la divergence existant entre la facilité de manipulation de l'information texte d'une part et graphique d'autre part. C'est une indication de l'utilité des images que les livres et documents ne contiennent pas encore suffisamment même après toutes ces années. Il est clair que c'est une des propriétés principales des environnements interactifs qui sont explorées pour le moment.

1.4.2. La reconnaissance et le rappel

Apprendre à utiliser un système demande la mémorisation d'information. Une considération importante dans la création d'une interface est la nature du dialogue entre le système et son utilisateur. Le but est de minimiser la quantité d'information que l'utilisateur doit mémoriser pour dialoguer avec le système.

Il existe deux approches possibles pour créer les dialogues homme-machine. La première est le dialogue contrôlé par le système, dans laquelle le système affiche tout ce qui est utile pour une tâche et puis demande à l'utilisateur sa prochaine action. Le système guide l'utilisateur. Tous les systèmes "conventionnels" et orientés menus appartiennent à cette catégorie. L'autre approche est le dialogue contrôlé par l'utilisateur, dans laquelle l'utilisateur demande une action en donnant les commandes nécessaires. Ici, l'utilisateur doit se souvenir des commandes et leur ordre d'exécution pour accomplir une certaine tâche.

Pendant la pensée consciente, le cerveau utilise plusieurs niveaux de mémoire, le plus important étant la mémoire à court-terme. Beaucoup d'études

ont analysé la mémoire à court-terme et son rôle dans la pensée. Les conclusions suivantes en ressortent :

* la pensée consciente traite avec des concepts en mémoire à court-terme
* la capacité de la mémoire à court-terme est limitée.

Lorsque toute l'information pertinente est visible, l'affichage facilite le stockage dans la mémoire à court-terme en agissant comme un "cache visuel". Penser devient plus facile et plus productif. Un dialogue bien construit peut réellement améliorer la qualité de la pensée de l'utilisateur. La nature conversationnelle d'un dialogue contrôlé par le système implique une forme réellement interactive de communication. Cela permet au système de donner une rétroaction positive ou négative instantanée à l'utilisateur en réponse à son action précédente, en plus d'inciter l'utilisateur pour la prochaine étape.

Par contre, les dialogues contrôlés par l'utilisateur sont généralement plus faciles à implémenter et plus efficaces pour exécuter une tâche particulière une fois que l'utilisateur a quelque expérience. Ils sont d'habitude plus flexibles et des abréviations sont souvent disponibles pour exécuter une tâche particulière. L'ordre des étapes du dialogue n'est pas fixe et cela peut être plus efficace pour un utilisateur expérimenté bien que le contraire soit vrai pour un utilisateur occasionnel ou novice. Les dialogues contrôlés par l'utilisateur donnent une charge supplémentaire à la mémoire de l'utilisateur.

1.5. La psychologie et le comportement de l'utilisateur

Pour comprendre ce que l'utilisateur fait, il faut premièrement connaître son but (ce qu'il essaye de faire) puisque toutes ses actions sont organisées pour accomplir ce but. Nous ne pouvons pas comprendre son comportement si nous ne connaissons pas son but. Ensuite, nous devons connaître la structure de la tâche , les "règles du jeu" qui déterminent l'ensemble des actions qu'il peut ou ne peut pas accomplir. La partie la plus importante de la structure de la tâche dans une interaction homme-machine est l'interface avec l'utilisateur du système informatique. Il faut se préoccuper ensuite de la connaissance de l'utilisateur, car il ne peut pas exploiter la structure de la tâche sans une connaissance effective de celle-ci (connaître les règles du jeu ne suffit pas pour bien jouer). Finalement, l'utilisateur a des limites de traitement qui le forcent à adopter des stratégies lui permettant de surmonter ces limites (par exemple, l'homme a une mémoire à court terme et une tendance à commettre occasionnellement des erreurs).

Nous pouvons donc résumer le comportement de l'utilisateur comme suit :

le but de l'utilisateur +

la structure de la tâche +

la connaissance de l'utilisateur +

les limites de traitement de l'utilisateur ==>

le comportement de l'utilisateur.

Des concepts ont été empruntés à différents domaines tels que la psychologie cognitive et la linguistique pour composer des modèles du comportement de l'utilisateur. De bons modèles du comportement de l'utilisateur peuvent aider à prédire les réactions de celui-ci face à de nouvelles interfaces et à déduire des règles pratiques de conception.

Des expériences intéressantes ont été menées pour déterminer les effets de différents facteurs sur l'utilisateur, sa satisfaction et sa productivité. Des facteurs tels que la disposition de l'écran, la taille des menus, l'aspect de la souris, l'utilisation de couleurs, et le temps de réponse de l'ordinateur ont été analysés pour déterminer leur influence sur l'utilisateur, à un niveau assez bas (son temps de réponse, son taux d'erreurs, par exemple), l'influence de la forme de l'interface sur les aspects de haut niveau des performances de l'utilisateur tels que le temps d'apprentissage, le temps nécessaire pour effectuer certaines tâches et l'organisation du travail de l'utilisateur a aussi été étudiée et de nouvelles formes d'interaction homme-machine sont analysées.

Le concepteur considère souvent l'interface comme le terminal (hardware) plus le software qui reçoit, interprète et renvoit les messages à l'utilisateur. Cette définition peut satisfaire le concepteur et le programmeur mais elle est inadéquate du point de vue de l'utilisateur.

Psychologiquement, l'interface est n'importe quelle partie du système avec laquelle l'utilisateur entre en contact, soit physiquement, perceptuellement ou conceptuellement. En fait, l'utilisateur a souvent connaissance de plus d'éléments que ce que le concepteur de l'interface aurait désiré qu'il connaisse. Un exemple traditionnel est le message d'erreur pour un overflow d'un registre interne. Un autre exemple est celui où l'utilisateur apprend à rafraîchir l'écran pour accélérer le système parce que cette commande réorganise les structures de données. Dans les deux cas, l'utilisateur est conscient, bien que vaguement, de ces structures internes qui deviennent de ce fait partie intégrante de l'interface. L'utilisateur développe une vue conceptuelle du système à partir du comportement général du système.

L'organisation conceptuelle du système, du point de vue de l'utilisateur (le modèle conceptuel du système pour l'utilisateur) est partie intégrante de l'interface. Le modèle conceptuel est l'organisation de travail du système, la manière dont il doit être

utilisé pour accomplir certaines tâches. Il doit être enseigné à l'utilisateur et doit être renforcé par le comportement du système. Il s'en suit que l'interface comprend non seulement le comportement de l'utilisateur devant son terminal mais aussi l'apprentissage et la documentation.

Un modèle conceptuel de l'utilisateur est la vue qu'il a du système, ce qui lui permet de comprendre et d'interagir avec celui-ci. Le développement d'un modèle conceptuel convenable est une étape très importante dans la création d'un langage d'interaction.

L'interface homme-machine est donc plus qu'un simple composant à rajouter à la fin de la conception du système. Elle le pénètre profondément. Elle doit donc être prise en considération très tôt dans le développement d'un système.

1.6. Conclusion

Après avoir présenté les principaux standards graphiques (GKS, PHIGS, CGM et CGI), nous avons analysé certains moyens graphiques existants (la souris, les menus, les fenêtres, ...) en les introduisant dans la communication homme-machine. Nous avons expliqué comment et en quoi ils peuvent rendre cette communication beaucoup plus agréable pour l'utilisateur. Nous n'avons relevé que quelques moyens, ceux-ci sont généralement utilisés par la plupart des systèmes. Ils nous ont permis de situer le contexte dans lequel nous travaillons.

Nous avons enfin proposé quelques aspects de la psychologie de l'utilisateur, ceux qui interfèrent avec son comportement vis-à-vis des systèmes graphiques. De toute cette présentation, nous avons déduit que même si ces moyens graphiques amélioraient généralement l'interface, ils peuvent distraire et gêner l'utilisateur.

Il est donc indispensable d'utiliser un style cohérent d'interaction avec la souris, les menus et les fenêtres se chevauchant. Cette cohérence a l'avantage d'être apprise une fois pour toutes mais a l'inconvénient d'obliger toutes les fonctions à partager les mêmes concepts et mécanismes de saisie et de restitution.

Il est à noter que certaines personnes préconisent la reconnaissance du langage parlé comme étant la meilleure technique de saisie. Mais même si nous avions cette capacité, ce ne serait pas une interface idéale pour beaucoup de produits. La complexité d'une interface devrait idéalement refléter la complexité de la tâche à spécifier ou à contrôler. Le langage naturel est un outil de communication complexe et incommode.

C'est une telle perte de temps et souvent tellement difficile pour nous d'exprimer ce que nous voulons, que nous désignons plus facilement quelque chose au lieu de le décrire. Désigner est plus rapide, plus simple et moins ambigu. Quand des personnes

discutent de choses simples, elles ont souvent besoin de répétition avant que chacun ne pense avoir compris. Et même, elles se trompent encore souvent.

Chapitre 2. La place du traitement graphique dans un processus de spécification

2.1. Introduction

Après avoir présenté les fonctionnalités du traitement graphique , nous allons tenter de définir les besoins de graphiques dans un éditeur de saisie et de restitution pour un langage de spécifications conceptuelles, et en particulier pour le langage D.S.L.

Nous commencerons ce chapitre en rappelant la méthodologie de travail d'un analyste et les différentes définitions s'y rapportant. Toutes les informations et définitions reprises ici seront largement inspirées de [BODA83]. Ensuite, nous essayerons de définir en quoi le traitement graphique peut aider l'analyste dans l'interprétation de son analyse.

Nous ne formulerons qu'un ensemble d'hypothèses à ce sujet, une étude approfondie étant impossible à réaliser dans le cadre de ce mémoire.

Sur base d'un exemple extrait du schéma de la dynamique, nous présenterons les améliorations pouvant être apportées aux rapports de synthèse qui existent actuellement.

2.2. Méthodologie de travail pour l'analyste

Le modèle général de tout système d'information peut se présenter comme suit :

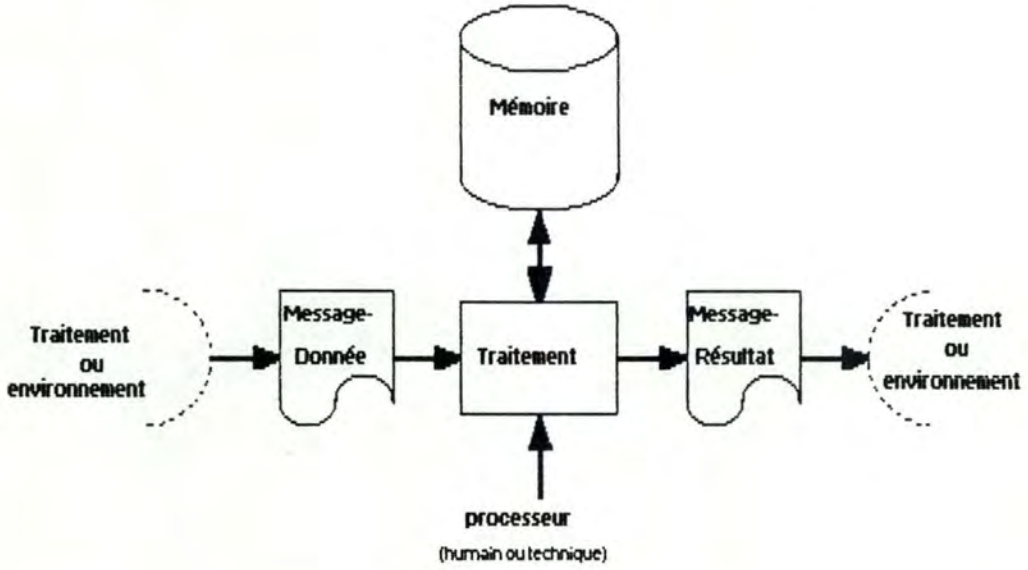


Figure 2.1.

Un message-donnée, en provenance d'un traitement ou de l'environnement du système d'information, est communiqué au système d'information, qui le traite via un processeur, éventuellement à l'aide de sa mémoire, pour engendrer un message-résultat lequel est à son tour transmis à un autre traitement ou à l'environnement du système d'information.

Par rapport à ce modèle général, nous pouvons reprendre les définitions de cinq modèles particuliers.

Le modèle de STRUCTURATION DES INFORMATIONS sert à définir la sémantique des données appartenant à la mémoire ou base des informations du système d'information. La structuration des informations porte notamment sur la définition des données et des relations entre celles-ci, sur l'analyse de leurs conditions d'existence et

des valeurs qu'elles peuvent prendre. Ce modèle est aussi utilisé pour définir les messages qui véhiculent les informations.

Le modèle de STRUCTURATION DES TRAITEMENTS doit permettre la décomposition, par raffinements successifs, d'un traitement global en traitements de plus en plus élémentaires.

Le modèle de la DYNAMIQUE DES TRAITEMENTS complète le modèle de structuration des traitements en permettant de décrire les enchaînements entre ceux-ci.

Le modèle de la STATIQUE DES TRAITEMENTS a pour but, d'une part, de préciser pour un traitement donné, les messages-données et la partie de la mémoire du système d'information nécessaires à l'obtention des messages-résultats et d'autre part, de spécifier, sous une forme adéquate, la procédure de traitement qui assure la transformation, à l'aide de la base de données du système d'information, des messages-données en messages-résultats.

Le modèle des RESSOURCES sert à caractériser le comportement des PROCESSEURS qui exécutent les procédures de traitement. Ce dernier modèle sera notamment utilisé pour spécifier les modalités d'utilisation des ressources telles que les services, les hommes, les équipements et les moyens financiers ainsi que pour caractériser les capacités d'activité des processeurs et leurs calendriers de disponibilité.

Au vu de ces modèles d'aide à l'analyse fonctionnelle détaillée, nous pouvons en déduire une méthodologie.

Celle-ci se décompose en deux parties principales : d'une part l'analyse des données et d'autre part l'analyse des traitements. L'analyste peut aborder ces deux parties parallèlement ou dans un ordre quelconque.

L'analyse des données se résume au modèle de structuration des informations (modèle E.R.A. : Entities-Relationships-Attributes). L'analyse des traitements comprend trois phases : la modélisation de structuration des traitements, la modélisation de la dynamique des traitements et la modélisation de la statique des traitements. Il est préférable d'effectuer ces trois étapes dans cet ordre.

2.3. Les apports du traitement graphique dans cette méthodologie

Actuellement, il n'existe qu'un éditeur texte pour aider l'analyste dans la saisie de ses données. Cet éditeur est basé sur le langage de spécifications D.S.L.

Le langage D.S.L. (Dynamic Specification Language) appartient à la famille des langages P.S.L. (Problem Statement Language). Il présente les caractéristiques suivantes [BODA83] :

C'est un langage de description des spécifications d'un système d'information.

Il est non procédural au sens où il permet la spécification dans un ordre quelconque et de manière progressive du système d'information.

Il est basé sur le modèle E.R.A. En effet, il est construit à partir des notions suivantes :

- des types d'OBJET dont le nombre est fixé par le langage
- des types de RELATION entre ces objets, dont le nombre est également limité
- des PROPRIETES associées à ces types d'objet ou de relation.

L'éditeur texte permet à présent à l'utilisateur d'effectuer une saisie de ses données sans qu'il n'ait à connaître la syntaxe du langage.

Pour chaque modèle utilisé par l'analyste, il existe un support de représentation graphique. Nous supposons que lors de la conception de son système d'information, l'analyste utilise ces diagrammes comme première approche. S'il a l'occasion d'utiliser un éditeur graphique, il ne devra donc plus traduire sa spécification en phrases D.S.L. L'opération d'encodage ne sera pas nécessairement plus rapide, mais probablement plus agréable puisqu'elle se fera via les graphiques et non plus uniquement via le texte. L'analyste aura de plus une rétroaction immédiate de ses modèles, ce qu'il n'avait pas avec l'éditeur texte en saisie, pour autant qu'il ne dispose pas d'un outil de restitution.

De plus, pour découvrir des erreurs éventuelles, il doit soit relire tout un ensemble de rapports D.S.L, soit reproduire ses modèles en se basant sur ces rapports, opération fastidieuse et très coûteuse en temps. Grâce à l'éditeur graphique, il aurait une rétroaction immédiate de ses modèles et la possibilité d'en imprimer le squelette indispensable à une meilleure visualisation.

Méfions-nous tout de même d'une conclusion trop hâtive, car le rapport graphique n'est pas assez complet pour que le concepteur puisse effectuer une analyse détaillée de ses spécifications. De plus, par l'utilisation fréquente d'un éditeur graphique, il pourrait ne plus maîtriser le langage D.S.L et donc éprouver des problèmes lors de l'interprétation des rapports existants. Il faudrait alors songer une représentation externe de la documentation indépendante de D.S.L.

Un autre avantage de l'éditeur graphique serait l'obtention rapide et automatique de diagrammes indispensables pour une meilleure présentation du futur système aux utilisateurs. Cet avantage nécessite cependant une étape de consolidation. Si l'analyste a encodé son diagramme petit à petit, ou si une partie a été saisie via l'éditeur texte, le système devra posséder des outils lui permettant de constituer un schéma global lisible et esthétique. Les diagrammes qu'il obtiendra seront propres, clairs et nets. Il ne devra plus les présenter dessinés à main levée ou à la latte. C'est donc ici un gain de temps appréciable pour tout le monde et un gain de productivité pour l'analyste. Grâce aux diagrammes, la discussion avec les futurs utilisateurs sera beaucoup plus facile et profitable. Le processus itératif de mise au point du modèle du système d'information par corrections successives sera beaucoup plus rapide puisque l'analyste pourra répercuter les modifications à apporter aux modèles immédiatement via l'éditeur graphique. Il pourra donc ainsi présenter les modifications successives aux utilisateurs à l'aide de diagrammes mis à jour proprement et non plus griffonnés.

Avant de pouvoir aborder la conception d'un éditeur graphique, il est indispensable d'étudier la démarche de l'analyste face à cet éditeur.

Soit, il l'utilisera comme support à sa réflexion. C'est-à-dire qu'il esquissera son schéma devant son terminal et procédera par essais-erreurs. Il est clair qu'un tel éditeur ne peut comporter de pilote. L'analyste devra être libre de faire ce qu'il veut, car il n'a pas d'idée a priori du diagramme qu'il désire obtenir. Des contrôles immédiats sont alors nécessaires pour éviter que son schéma ne devienne incohérent ou ne reste incomplet.

Soit, il utilisera l'éditeur comme support de saisie. Dans ce cas, il commencera son analyse sur papier pour ensuite l'encoder. Ici, puisque l'analyste sait a priori ce qu'il désire dessiner, le système doit le piloter. Ceci implique des contrôles de cohérence et de complétude implicite dans la saisie.

A l'heure actuelle, il n'existe pas d'étude de type ergonomique permettant de dire si un éditeur de saisie est un outil qui aide l'analyste à concevoir son modèle et donc à penser, ou simplement un outil de support graphique. Nous estimons que tout bon analyste, comme tout bon programmeur, doit commencer son travail sur un support papier et non avec un éditeur, quelle qu'en soit sa qualité.

2.4. Illustration sur base de rapports existants

A partir d'un exemple simple du modèle de la dynamique, nous avons produit un ensemble de rapports via l'analyseur D.S.A. (Dynamic Specification Analyser). Nous les avons tous regroupés en fin de cette section. Le premier de ces rapports est produit à la saisie des données et présente l'exemple.

Outre l'intégration des spécifications dans la base de données, l'analyseur D.S.A. permet de produire des rapports documentaires de deux types [BODA83].

La première catégorie concerne les objets de la base de données pris individuellement. Ces rapports fournissent des renseignements généraux tels que le nombre d'objets d'un type donné possédant des synonymes, par exemple. Ils permettent également la restitution des spécifications relatives à un objet donné, aussi bien celles décrites directement par l'analyste que celles déduites de la définition de leur forme complémentaire. Même avec un éditeur graphique, ces rapports restent indispensables pour une documentation complète des objets définis.

La deuxième catégorie documente des structures enregistrées dans la base de données des spécifications. Ces rapports décrivent des relations entre objets de types différents. Ils sont les plus susceptibles d'être améliorés par un éditeur graphique.

Dans notre exemple, nous présentons trois types de rapports. Les rapports "STRUCTURE" et "STRUCURED FORMATED STATEMENT REPORT" présentent sous forme de listes "à libellés décalés" une structure associée à un ou plusieurs objets D.S.L. C'est l'utilisateur qui spécifie les types d'objets et de relations qui définissent la structure souhaitée. Ces deux rapports reprennent les mêmes informations si ce n'est que la relation entre les objets est plus complète dans le rapport "STRUCTURED FORMATED STATEMENT REPORT".

Le rapport "EXTENDED PICTURE" est similaire dans son contenu au rapport "STRUCTURE". Il présente les mêmes structures sous la forme dessinée d'une arborescence.

Les paramètres que nous avons définis pour chacun de ces rapports sont les suivants.

Tous les objets peuvent être considérés : OBJECTS = ALL.

Toutes les relations de la dynamique descendante doivent être représentées sur le rapport : RELATION = dwn-dynamics.

Pour chacun de ces rapports, les objets de départ sont les trois messages. Leurs noms sont spécifiés dans le fichier "MESS.NAM" : FILE = mess.nam.

Il est évident que ces trois rapports ne sont qu'un échantillon des possibilités de l'analyseur D.S.A. De nombreux autres rapports peuvent être produits, et de nombreux paramètres peuvent être spécifiés de manière à extraire les parties de la base de données intéressant directement le concepteur.

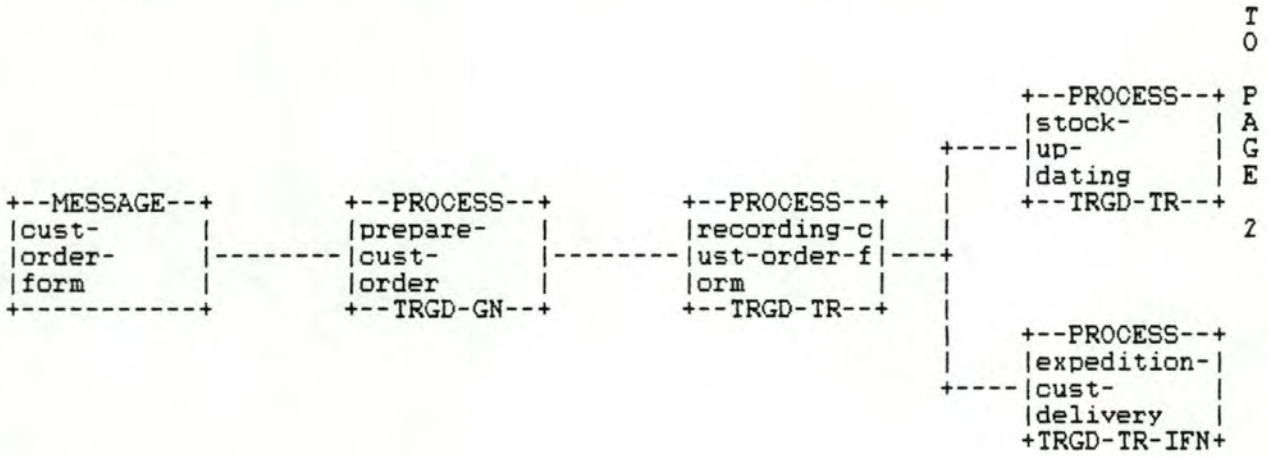
A la suite de ces rapports, nous présentons le diagramme contenant les mêmes informations qui pourraient être produit par l'éditeur graphique. Le lecteur trouvera dans [LEFE87] un algorithme de production automatique d'un tel schéma.

```
Interactive Design Approach    IDA2.0          Aug 18, 1987 17:24:51 Page 1
                            FNDP - Namur VAX/VMS

                        Input Processor Source Listing

  Parameters:   LANGUAGE=DSL  DB=chap3.dbf  INPUT=chap3.dyn  SOURCE-LISTING
    NOCROSS-REFERENCE  UPDATE  DATA-BASE-REFERENCE  NOAPPEND-TEXT

    1
    2 DEFINE  MESSAGE cust-order-form;
    3         ON GENERATION TRIGGERS  prepare-cust-order;
    4
    5 DEFINE  MESSAGE                 selection-supplying-product;
    6         ON GENERATION TRIGGERS  recording-supplying-product;
    7
    8 DEFINE  MESSAGE                 selection-exhausted-product;
    9         ON GENERATION TRIGGERS  recording-exhausted-product;
   10
   11 DEFINE  PROCESS                 prepare-cust-order;
   12         ON TERMINATION TRIGGERS recording-cust-order-form
   13                                 IF identification-cust-possible;
   14
   15 DEFINE  PROCESS                 recording-cust-order-form;
   16         ON TERMINATION TRIGGERS stock-up-dating
   17                                 IF valid-cust-order;
   18         ON TERMINATION TRIGGERS expedition-cust-delivery
   19                                 IF NOT valid-cust-order;
   20
   21 DEFINE  PROCESS                 recording-supplying-product;
   22         ON TERMINATION TRIGGERS stock-up-dating
   23                                 FOR EACH different-cust-order;
   24
   25 DEFINE  PROCESS                 stock-up-dating;
   26         ON TERMINATION TRIGGERS search-in-store
   27                                 IF product-requisition-possible;
   28         ON TERMINATION TRIGGERS expedition-cust-delivery
   29                                 IF NOT product-requisition-possible;
   30
   31 DEFINE  PROCESS                 search-in-store;
   32         ON TERMINATION TRIGGERS package-establishment;
   33
   34 DEFINE  PROCESS                 package-establishment;
   35         ON TERMINATION TRIGGERS expedition-cust-delivery;
   36
   37 DEFINE  PROCESS                 recording-exhausted-product;
   38         ON TERMINATION TRIGGERS expedition-cust-delivery
   39                                 FOR EACH different-cust-order;
   40
   41 DEFINE  PROCESS                 expedition-cust-delivery;
   42
   43 DEFINE  CONDITION               identification-cust-possible;
   44
   45 DEFINE  CONDITION               valid-cust-order;
   46
   47 DEFINE  CONDITION               product-requisition-possible;
   48
   49 DEFINE  SYSTEM-PARAMETER        different-cust-order;
   50
```

```
Interactive Design Approach     IDA2.0              Aug 18, 1987 17:24:51 Page 2
                              FNDP - Namur VAX/VMS

                        Input Processor Source Listing


50 lines with 27 statements.
No diagnostics.
```

```
Interactive Design Approach     IDA2.0          Aug 19, 1987 14:53:03 Page 1
                              FNDP - Namur VAX/VMS

                                Extended Picture

  Parameters:   LANGUAGE=DSL  DB=chap3.dbf  FILE=mess.nam  OBJECTS=ALL
    RELATIONS=dwn-dynamics  LINKS=50  PRINT  BOXES  ALL  HORIZONTAL-BOXES=4
    VERTICAL-BOXES=8  NOARROWS  NOPLOT  NOPUNCH  DUPLICATE-HORZ-BOXES
    NODUPLICATE-VERT-BOXES
```

```
Interactive Design Approach     IDA2.0          Aug 19, 1987 14:53:03 Page 2
                              FNDP - Namur VAX/VMS

                                Extended Picture

NAME=cust-order-form                                  PAGE 1 OF 2
```



```
Interactive Design Approach     IDA2.0            Aug 19, 1987 14:53:03 Page 3
                              FNDP - Namur VAX/VMS

                                Extended Picture

NAME=cust-order-form                                        PAGE 2 OF 2

F                          +--PROCESS--+        +--PROCESS--+         +--PROCESS--+
R                          |search-    |        |package-est|         |expedition-|
O                    +-----|in-        |--------|ablishment |---------|cust-      |
M +--PROCESS--+      |     |store      |        |           |         |delivery   |
  |stock-     |      |     +--TRGD-TR--+        +--TRGD-TR--+         +--TRGD-TR--+
P |up-        |------+
A |dating     |      |     +--PROCESS--+
G +--TRGD-TR--+      |     |expedition-|          Names occurs
E                    +-----|cust-      |          elsewhere
                           |delivery   |
1                          +TRGD-TR-IFN+

  +--PROCESS--+
  |expedition-|            Names occurs
  |cust-      |            elsewhere
  |delivery   |
  +TRGD-TR-IFN+
```

```
Interactive Design Approach     IDA2.0            Aug 19, 1987 14:53:03 Page 4
                              FNDP - Namur VAX/VMS

                                Extended Picture

NAME=selection-supplying-product                            PAGE 1 OF 2

                                                                  +--PROCESS--+ T
                                                                  |search-    | O
                                                            +-----|in-        |
  +--MESSAGE--+          +--PROCESS--+        +--PROCESS--+|     |store      | P
  |selection- |          |recording- |        |stock-     ||     +--TRGD-TR--+ A
  |supplying- |----------|supplying- |--------|up-        |+-----+            G
  |product    |          |product    |        |dating     ||     +--PROCESS--+ E
  +-----------+          +--TRGD-GN--+        +--TRGD-TR--+|     |expedition-|
                                                            +-----|cust-      | 2
                                                                  |delivery   |
                                                                  +TRGD-TR-IFN+
```

```
Interactive Design Approach     IDA2.0              Aug 19, 1987 14:53:03 Page 5
                              FNDP - Namur VAX/VMS

                                  Extended Picture

NAME=selection-supplying-product                          PAGE 2 OF 2

F +--PROCESS--+         +--PROCESS--+         +--PROCESS--+
R |search-    |         |package-est|         |expedition-|
O |in-        |---------|ablishment |---------|cust-      |
M |store      |         |           |         |delivery   |
  +--TRGD-TR--+         +--TRGD-TR--+         +--TRGD-TR--+
P
A +--PROCESS--+
G |expedition-|          Names occurs
E |cust-      |          elsewhere
  |delivery   |
1 +TRGD-TR-IFN+
```

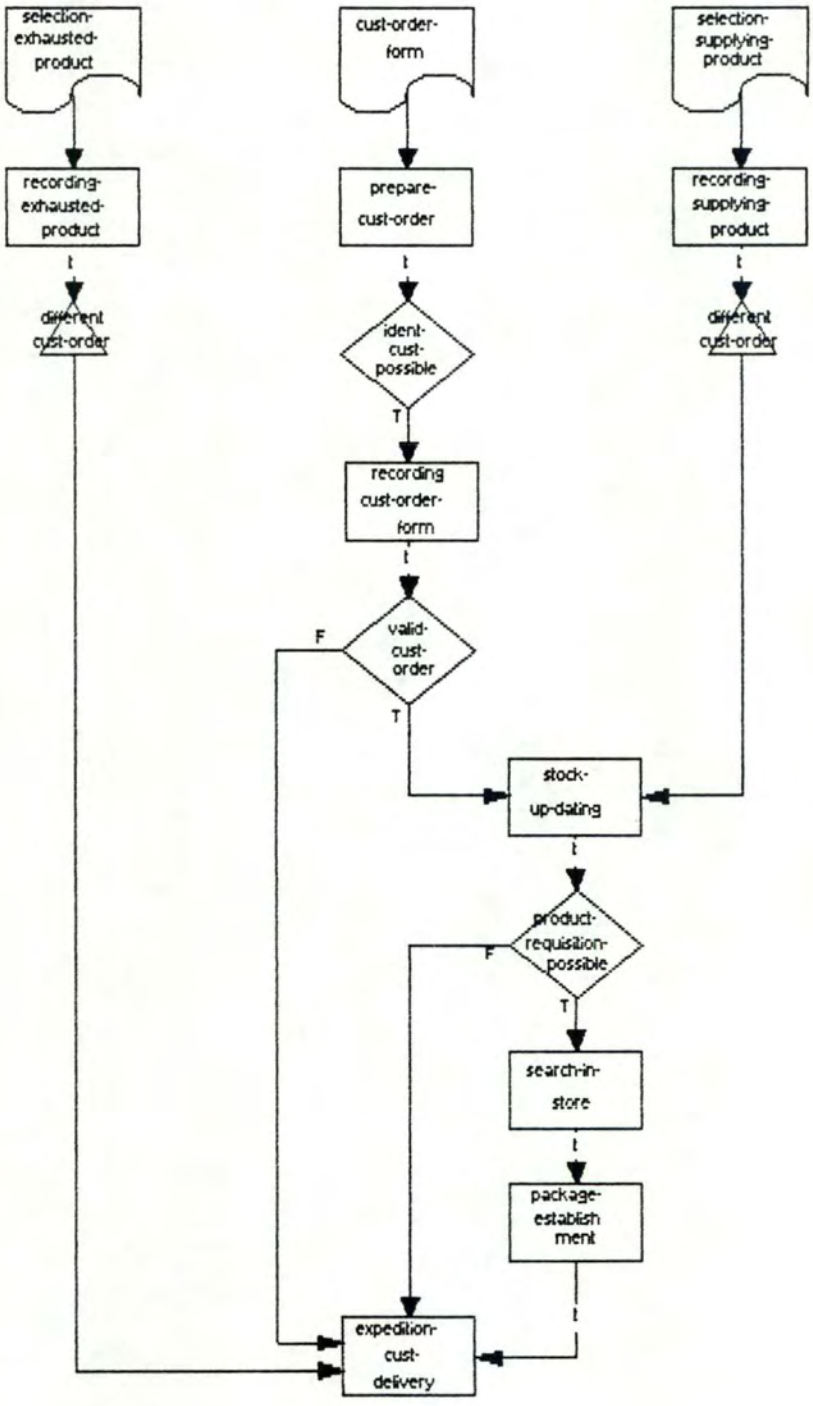
```
Interactive Design Approach     IDA2.0              Aug 19, 1987 14:53:03 Page 6
                              FNDP - Namur VAX/VMS

                                  Extended Picture

NAME=selection-exhausted-product                          PAGE 1 OF 1

  +--MESSAGE--+         +--PROCESS--+         +--PROCESS--+
  |selection- |         |recording- |         |expedition-|
  |exhausted- |---------|exhausted- |---------|cust-      |
  |product    |         |product    |         |delivery   |
  +-----------+         +--TRGD-GN--+         +--TRGD-TR--+
```

```
Interactive Design Approach    IDA2.0          Aug 19, 1987 14:53:39 Page 7
                            FNDP - Namur VAX/VMS

                                Structure

 Parameters:   LANGUAGE=DSL  DB=chap3.dbf  FILE=mess.nam  NOPUNCH
   OBJECTS=ALL  RELATIONS=dwn-dynamics  PRINT  INDENT=2  LEVELS=50
   TYPES-MARGIN=36  RELATIONS-MARGIN=48  LINE-NUMBERS  LEVEL-NUMBERS
   STATISTICS  NONEW-PAGE  OBJECT-TYPES  ROW-ORDER=STANDARD
   COLUMN-ORDER=STANDARD  NOLOWEST-LEVEL-ONLY  NORELATION-MATRIX
   NOCOMPRESS-RELATION-MATRIX



  1   1 cust-order-form              MESSAGE
  2    2  prepare-cust-order         PROCESS      (TRIGGERED-BY-GEN)
  3     3   recording-cust-order-form
                                     PROCESS      (TRIGGERED-BY-TERM[-IF])
  4      4    stock-up-dating        PROCESS      (TRIGGERED-BY-TERM[-IF])
  5       5     expedition-cust-delivery
                                     PROCESS      (TRIGGERED-BY-TERM-IF-NOT)
  6       5     search-in-store      PROCESS      (TRIGGERED-BY-TERM[-IF])
  7        6      package-establishment
                                     PROCESS      (TRIGGERED-BY-TERM[-IF])
  8         7       expedition-cust-delivery
                                     PROCESS *    (TRIGGERED-BY-TERM[-IF])
  9      4    expedition-cust-delivery
                                     PROCESS *    (TRIGGERED-BY-TERM-IF-NOT)

LEVEL COUNT   LEVEL COUNT   LEVEL COUNT   LEVEL COUNT   LEVEL COUNT
    1     1       2     1       3     1       4     2       5     2
    6     1       7     1


  1   1 selection-supplying-product
                                     MESSAGE
  2    2  recording-supplying-product
                                     PROCESS      (TRIGGERED-BY-GEN)
  3     3   stock-up-dating          PROCESS      (TRIGGERED-BY-TERM[-IF])
  4      4    search-in-store        PROCESS      (TRIGGERED-BY-TERM[-IF])
  5       5     package-establishment
                                     PROCESS      (TRIGGERED-BY-TERM[-IF])
  6        6      expedition-cust-delivery
                                     PROCESS      (TRIGGERED-BY-TERM[-IF])
  7      4    expedition-cust-delivery
                                     PROCESS *    (TRIGGERED-BY-TERM-IF-NOT)

LEVEL COUNT   LEVEL COUNT   LEVEL COUNT   LEVEL COUNT   LEVEL COUNT
    1     1       2     1       3     1       4     2       5     1
    6     1


  1   1 selection-exhausted-product
                                     MESSAGE
  2    2  recording-exhausted-product
```

```
Interactive Design Approach     IDA2.0              Aug 19, 1987 14:53:39 Page 8
                              FNDP - Namur VAX/VMS

                                    Structure

                                     PROCESS      (TRIGGERED-BY-GEN)
  3     3   expedition-cust-delivery
                                     PROCESS      (TRIGGERED-BY-TERM[-IF])

 LEVEL COUNT   LEVEL COUNT   LEVEL COUNT
     1     1       2     1       3     1
```

```
Interactive Design Approach    IDA2.0          Aug 19, 1987 14:54:05 Page 9
                            FNDP - Namur VAX/VMS

                    Structured Formated Statement Report

 Parameters:   LANGUAGE=DSL  DB=chap3.dbf  FILE=mess.nam  NOPUNCH
   OBJECTS=ALL  RELATIONS=dwn-dynamics  PRINT  INDENT=2  LEVELS=50
   TYPES-MARGIN=36  RELATIONS-MARGIN=48  LINE-NUMBERS  LEVEL-NUMBERS
   STATISTICS  NONEW-PAGE  OBJECT-TYPES  ROW-ORDER=STANDARD
   COLUMN-ORDER=STANDARD  NOLOWEST-LEVEL-ONLY  NORELATION-MATRIX
   NOCOMPRESS-RELATION-MATRIX



 1   1 cust-order-form           (MESSAGE)
 2    2  prepare-cust-order      (PROCESS)
                                          TRIGGERED BY GENERATION OF
                                          cust-order-form
 3     3   recording-cust-order-form
                                 (PROCESS)
                                          TRIGGERED BY TERMINATION OF
                                          prepare-cust-order IF
                                          identification-cust-possible
 4      4    stock-up-dating     (PROCESS)
                                          TRIGGERED BY TERMINATION OF
                                          recording-cust-order-form IF
                                          valid-cust-order
 5       5     expedition-cust-delivery
                                 (PROCESS)
                                          TRIGGERED BY TERMINATION OF
                                          stock-up-dating IF NOT
                                          product-requisition-possible
 6       5     search-in-store   (PROCESS)
                                          TRIGGERED BY TERMINATION OF
                                          stock-up-dating IF
                                          product-requisition-possible
 7        6      package-establishment
                                 (PROCESS)
                                          TRIGGERED BY TERMINATION OF
                                          search-in-store
 8         7       expedition-cust-delivery
                                 (PROCESS) *
                                          TRIGGERED BY TERMINATION OF
                                          package-establishment
 9      4    expedition-cust-delivery
                                 (PROCESS) *
                                          TRIGGERED BY TERMINATION OF
                                          recording-cust-order-form IF NOT
                                          valid-cust-order

LEVEL COUNT   LEVEL COUNT   LEVEL COUNT   LEVEL COUNT   LEVEL COUNT
    1     1       2     1       3     1       4     2       5     2
    6     1       7     1
```

```
Interactive Design Approach    IDA2.0          Aug 19, 1987 14:54:05 Page 10
                          FNDP - Namur VAX/VMS

                    Structured Formated Statement Report

  1   1 selection-supplying-product
                                  (MESSAGE)
  2    2  recording-supplying-product
                                  (PROCESS)
                                                TRIGGERED BY GENERATION OF
                                                selection-supplying-product
  3     3   stock-up-dating       (PROCESS)
                                                TRIGGERED BY TERMINATION OF
                                                recording-supplying-product FOR
                                                EACH different-cust-order
  4      4    search-in-store     (PROCESS)
                                                TRIGGERED BY TERMINATION OF
                                                stock-up-dating IF
                                                product-requisition-possible
  5       5     package-establishment
                                  (PROCESS)
                                                TRIGGERED BY TERMINATION OF
                                                search-in-store
  6        6      expedition-cust-delivery
                                  (PROCESS)
                                                TRIGGERED BY TERMINATION OF
                                                package-establishment
  7      4    expedition-cust-delivery
                                  (PROCESS) *
                                                TRIGGERED BY TERMINATION OF
                                                stock-up-dating IF NOT
                                                product-requisition-possible

LEVEL COUNT   LEVEL COUNT   LEVEL COUNT   LEVEL COUNT   LEVEL COUNT
    1     1       2     1       3     1       4     2       5     1
    6     1


  1   1 selection-exhausted-product
                                  (MESSAGE)
  2    2  recording-exhausted-product
                                  (PROCESS)
                                                TRIGGERED BY GENERATION OF
                                                selection-exhausted-product
  3     3   expedition-cust-delivery
                                  (PROCESS)
                                                TRIGGERED BY TERMINATION OF
                                                recording-exhausted-product FOR
                                                EACH different-cust-order

LEVEL COUNT   LEVEL COUNT   LEVEL COUNT
    1     1       2     1       3     1
```


selection-exhausted-product
cust-order-form
selection-supplying-product
recording-exhausted-product
prepare-cust-order
recording-supplying-product
different cust-order
ident-cust-possible
different cust-order
T
recording cust-order-form
valid-cust-order
F
T
stock-up-dating
product-requisition-possible
F
T
search-in-store
package-establish ment
expedition-cust-delivery

2.5. Conclusion

Nous avons rappelé que dans la méthodologie IDA ainsi que dans bien d'autres méthodologies, les différents aspects de la spécification sont supportés par une représentation graphique. Celle-ci permet une visualisation souvent simple des objets et relations de base de la spécification.

On constate généralement que les analystes procèdent à l'ébauche de leurs spécifications à l'aide de cette représentation graphique.

L'objectif d'un éditeur de saisie consistera dès lors à permettre à l'analyste d'encoder les résultats de son travail dans cette représentation graphique et de ne pas devoir la transformer en une représentation textuelle.

Nous avons exposé au cours de ce chapitre les avantages possibles que l'on peut attacher à une telle démarche et nous en avons souligné les éventuelles limites.

Nous avons constaté également que l'éditeur de saisie doit être complété par un éditeur de restitution, aussi bien pour présenter les informations saisies en mode texte que pour consolider celles qui ont été saisies en mode graphique.

Il reste à savoir comment l'analyste va réagir face à cet éditeur graphique. La question peut se résumer ainsi. "Cliquer" est-il égal à penser ou "cliquer" est-il égal à dessiner ?

Nous n'avions pas la prétention d'étudier en profondeur la méthodologie de l'analyste mais simplement d'introduire la question de manière à montrer que nous en sommes conscients. Il s'agirait en effet d'un travail de recherche d'une ampleur considérable qui ne constitue en aucun cas l'objectif de ce mémoire, lequel est en fait orienté vers la contribution à un éditeur graphique.

Chapitre 3. Etude d'un éditeur graphique de saisie

3.1. Introduction

L'éditeur graphique à concevoir dans le cadre du logiciel IDA se compose de deux parties : d'une part la saisie des données relevées par l'analyste dans l'entreprise, d'autre part la restitution de ces données ou de données qui auraient été saisies via l'éditeur texte avec des possibilités de mise à jour de celles-ci. Dans ce chapitre, nous allons analyser l'éditeur de saisie en deux étapes. Tout d'abord, nous allons étudier les fonctionnalités de logiciels existants, à savoir "Structured Architect" développé par ISDOS (Michigan) que nous avons eu l'occasion d'utiliser et de tester, et "I.E.W.", logiciel de chez Arthur Young Proware, dont nous avons vu une démonstration. Nous exposerons ensuite notre contribution au développement de l'éditeur graphique dédié à IDA. Nous proposerons une architecture de la partie saisie de cet éditeur et une possibilité de scénario. Nous soulèverons aussi les problèmes que nous avons rencontrés pour maintenir la cohérence entre les différentes bases de données du logiciel. Nous élargirons enfin cette prespective en présentant le projet sur lequel nous avons travaillé en collaboration avec l'université du Michigan : le SEEOF ("Software Engineering Environment of the Future") et en particulier la partie destinée à la communication homme-machine, le DES ("Display Edit Server").

3.2. Analyse de systèmes existants

3.2.1. Structured Architect

3.2.1.1. Le sujet

Structured Architect est un programme développé au Michigan par ISDOS, permettant la saisie graphique de "Structured Analysis" [MARC78], la traduction automatique de ce graphique en PSL ("Problem Statement Language") et la génération de toute une série de rapports. [S.A.86]

3.2.1.2. Structured Analysis

a. Présentation

"Structured Analysis" est basé sur les modèles suivants :

- Diagramme des flux de données
- Dictionnaire des données
- "Structured English"

"Structured Analysis" travaille avec une vue "top-down" du système à modéliser, c'est-à-dire une modélisation par raffinements successifs.

b. Définitions

- Le diagramme des flux de données décrit le cheminement des données à travers le système. Il est composé de processus qui transforment les données, de sources qui sont des objets extérieurs au système, générant ou recevant des données, de fichiers ou data stores où peuvent être stockées des données et enfin de flux de données ou Dataflows.

Leur représentation graphique est la suivante :

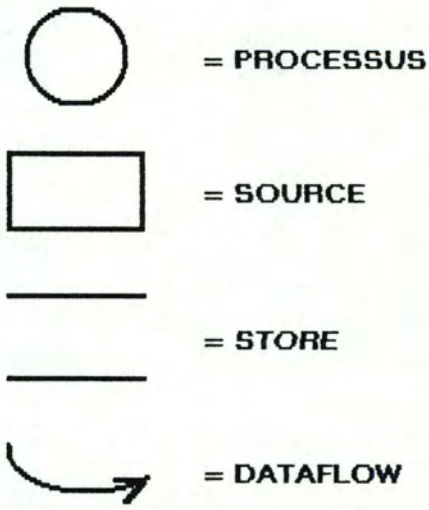


figure 3.1.

- Le dictionnaire des données fournit des définitions compréhensibles et précises pour les données et pour tous les composants du diagramme des flux.

- "Structured English" documente sous forme procédurale les transformations de données et le travail effectué sur ces données (documentation d'un processus).

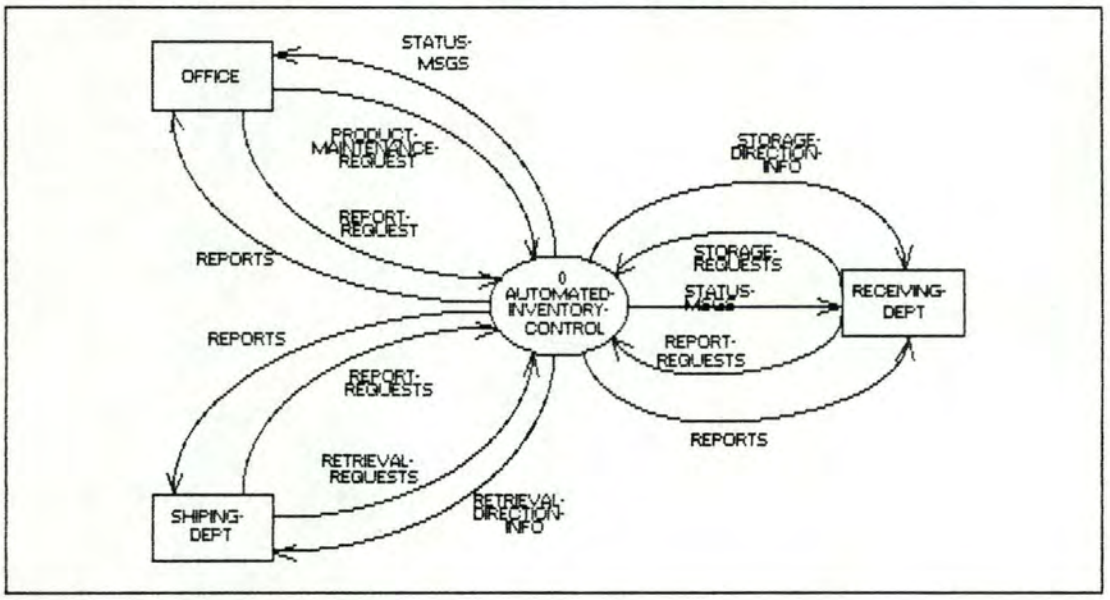
c. Restrictions du langage

* Au premier niveau (appelé CONTEXT), la définition d'un seul processus est autorisée. On ne peut pas y trouver de data store, et c'est le seul niveau qui accepte les sources. Celles-ci étant des objets extérieurs au modèle, elles sont donc indécomposables.

* A n'importe quel niveau, un processus pour lequel le "Structured English" a été défini est indécomposable.

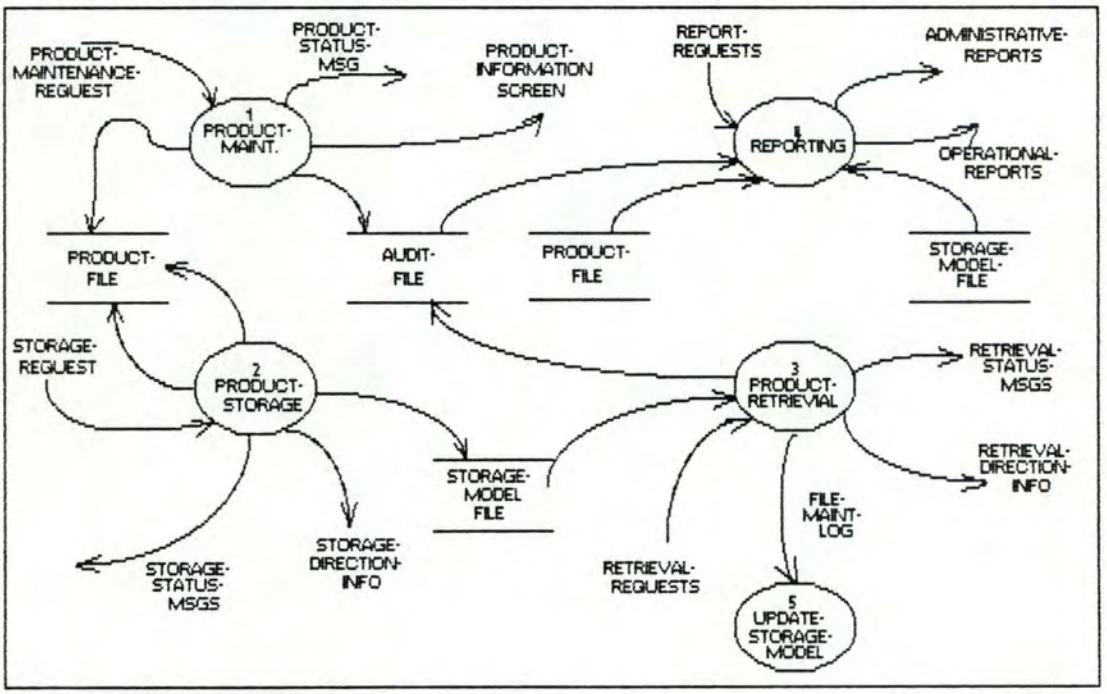
3.2.1.3. Le logiciel

Il est présenté sous forme de menus arborescents. Lors de l'édition d'un diagramme, le programme passe en mode graphique (figure 3.2.). Pour placer un objet graphique, l'utilisateur se positionne sur le type de celui-ci dans le menu et enfonce le bouton de la souris à l'endroit où il veut le placer dans la page graphique. L'identifiant de cet objet sera placé au centre de celui-ci. Pour placer un flux de données, l'utilisateur sélectionne l'option correspondante dans le menu et trace le chemin utilisé par la courbe en enfonçant le bouton de la souris aux différents points par lesquels il souhaite la voir passer; Pour descendre à un niveau de raffinement plus fin (décomposition d'un processus par exemple), il sélectionne l'option appropriée dans le menu, puis se positionne sur l'objet à décomposer et l'ouvre. Cette action a pour effet de lui offrir un nouvel écran graphique représentant un zoom de l'objet. (figure 3.3.). La décomposition d'un Dataflow, d'un Data store ou d'un Processus via Structured English fait passer l'utilisateur dans un éditeur texte plein écran lui permettant de remplir le dictionnaire des données. Chaque commande est accessible soit à partir de la souris, soit à partir du clavier (touches fonction et flèches). L'utilisateur est libre de créer son diagramme comme il le désire, dans l'ordre qu'il souhaite. Il n'y a donc aucune méthodologie sous-jacente17.



CONTEXT

figure 3.2.



AUTOMATED-INVENTORY-CONTROL 0

figure 3.3.

3.2.1.4. Critiques du logiciel

Lors de l'utilisation de Structured Architect; nous avons relevé certains aspects positifs que nous nous proposons d'introduire dans la conception de notre éditeur graphique, et d'autres, moins concluants, que nous essayerons d'éviter.

A chaque niveau et pour n'importe quel menu, l'utilisateur a la possibilité d'invoquer un écran d'aide, ce qui lui permet de se documenter au fur et à mesure de sa progression dans le logiciel. De plus, la même touche fonction est assignée à cet écran d'aide, quelque soit la situation de l'utilisateur dans le logiciel. Cette facilité se retrouve pour n'importe quelle fonction existant à plusieurs niveaux (exemple : "HELP" est toujours assigné à la touche fonction F2, "PRINT" à la touche F7, ...).

L'affichage des diagrammes et le rafraîchissement de l'écran s'effectuent rapidement.

Notons enfin que pour sélectionner un objet, il suffit à l'utilisateur d'être positionné dans une zone autour de celui-ci et non nécessairement de se trouver exactement sur cet objet (figure 3.5.).

En ce qui concerne les points négatifs, nous trouvons que les fonctions interdites à certains niveaux ne devraient pas y être accessibles (par exemple, la création d'un objet de type "data store" est interdite dans un graphique de type CONTEXT et pourtant l'option est disponible dans le menu).

Lors de la saisie de son diagramme, l'utilisateur pourrait avoir des difficultés à ne rien oublier. Des informations doivent être introduites dans divers sous-menus et rien ne lui indique ce qu'il a déjà fait et ce qui reste à faire étant donné qu'il n'existe pas de pilotage.

Lors de la création de son schéma, l'utilisateur a la possibilité de positionner deux Dataflows ou deux objets au même endroit. Les noms apparaissent donc l'un sur l'autre et le dessin devient illisible.

Exemple :

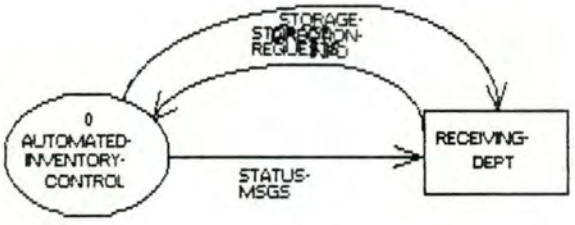


figure 3.4.

Lors de la suppression d'un processus, les Dataflows qui lui sont associés ne sont pas supprimés. Le logiciel pourrait proposer une suppression automatique.

Notons enfin que les objets sont tous de la même taille, et que celle-ci est constante.

3.2.1.5. Analyse du logiciel

Structured Architect a été développé à l'aide du logiciel de gestion graphique HALO. La seule façon pour saisir un modèle est la saisie graphique. La base de données graphique est donc toujours cohérente avec les spécifications. Il n'y a pas non plus de problèmes de concurrence, Structured Architect étant un logiciel mono-utilisateur. Le seul rôle de Structured Architect est la traduction automatique d'un diagramme en PSL.

1. Les objets (process, source, store).

   Les objets sont des entités statiques, c'est-à-dire des entités ayant une forme prédéfinie de taille fixe. A chaque occurrence d'un objet

sur une page graphique sont associés son nom et ses coordonnées. Le nom est centré par rapport à l'objet et aucune attention n'est portée sur le fait qu'il puisse dépasser des bornes de l'objet. Le nom peut être placé sur plusieurs lignes (maximum 3) si l'utilisateur le désire. Il devra alors insérer un caractère spécial (~) à l'endroit où il souhaite la césure.

Certains logiciels de saisie graphique se basent sur une grille, c'est-à-dire qu'ils ont une structure cachée, telle qu'une seule case de cette grille ne peut contenir qu'un seul objet. Cette méthode de travail permet un alignement automatique des objets et interdit le recouvrement de deux objets.

Structured Architect ne travaille pas à l'aide d'une telle grille. En effet, les objets peuvent être positionnés à n'importe quel endroit de l'espace graphique, un objet pouvant même en recouvrir un autre.

Pour la sélection, un objet est reconnu dès que le curseur se trouve en-deçà d'une distance fixe par rapport à son centre.

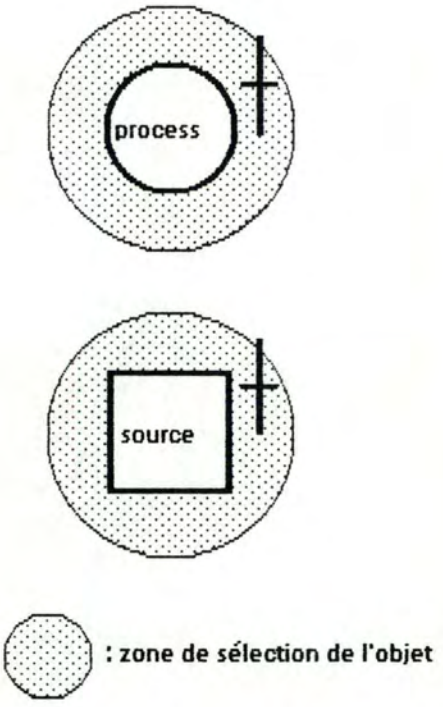


figure 3.5.

Dans la figure 3.5., l'objet sera sélectionné dans les deux cas, bien que le curseur (la croix) ne se trouve pas exactement sur lui.

Si deux objets sont proches l'un de l'autre et que l'utilisateur est dans une région de sélection commune aux deux objets, la sélection se fait sur celui dont le centre est le plus proche du curseur.

2. Les Dataflows.

Il s'agit de lignes reliant deux objets et passant par une série de point spécifiés par l'utilisateur. Il existe deux possibilités de représentation des Dataflows :

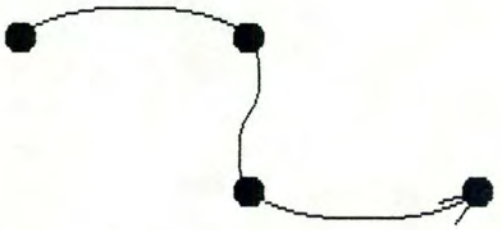
* AUTOCURVE = OFF (figure 3.6.) : tracé de segments de droite passant par les points spécifiés par l'utilisateur. Lors de la construction du Dataflow, les lignes suivent la souris jusqu'à ce que

l'utilisateur enfonce le bouton pour spécifier un point où une césure doit se faire. (méthode *Rubberband*)

AUTOCURVE = OFF

figure 3.6.

* AUTOCURVE = ON (figure 3.7.) : tracé d'une courbe passant par les points choisis. A la construction du Dataflow, l'utilisateur a toujours des segments de droite. C'est lorsqu'il a spécifié le point d'arrivée que la courbe est calculée



AUTOCURVE = ON

figure 3.7.

Le nom du Dataflow (label) est positionné à un endroit quelconque de la page graphique, là où l'utilisateur le désire. Pour la sélection d'un Dataflow, le curseur doit être positionné aux alentours de ce label (en-deçà d'une distance fixe).

La modification du tracé d'un Dataflow se fait soit en insérant un point de césure supplémentaire entre deux points, soit en en supprimant un, soit en en déplaçant un.

3. Le zoom.

L'agrandissement ou le rétrécissement du dessin se fait toujours par rapport au centre de l'écran, quelle que soit la position du curseur ou du dessin à ce moment. Le seul zoom possible est un zoom global (simple changement d'échelle). Il est impossible d'effectuer un zoom sur un objet (sauf pour descendre à un niveau de raffinement plus fin, auquel cas un nouvel écran graphique est proposé), et quel que soit la taille des objets, il n'y a pas de différence en ce qui concerne le raffinement des données affichées (Il n'y a jamais que l'identifiant des objets).

3.2.2. Information Engineering Workbench

3.2.2.1. Présentation du logiciel

"I.E.W." est un des rares logiciels d'aide à l'analyse conceptuelle actuellement sur le marché. Il possède un éditeur de saisie et un éditeur de restitution graphique permettant à l'utilisateur d'encoder son analyse de façon agréable. Il fonctionne sur micro-ordinateur (de type IBM PC/AT) dans un environnement multi-fenêtres. Ce logiciel a été développé à l'aide de GEM (gestionnaire de fenêtres et de menus similaire à MS-WINDOWS). Les seules informations que nous possédons sur ce logiciel sont extraites de fascicules publicitaires [IEWa] et [IEWb]. Les informations risquent donc de ne pas être entièrement objectives. Les remarques concernant la saisie des données (3.2.2.2.) et leur restitution (5.2.1.1.) ont pu être

formulées grâce à une démonstration du logiciel à laquelle nous avons eu l'occasion d'assister.

"I.E.W." est basé sur "Structured Analysis" [MARC78] étendu à quatre outils de saisie : le diagramme de décomposition, le diagramme des flux, le diagramme Entités-Relations et le diagramme d'actions.

Le diagramme de décomposition est l'instrument de base de l'analyse dans la méthodologie de "I.E.W.". Il représente une vue hiérarchisée de l'organisation (par raffinements successifs). Il est composé d'objets représentant les activités de l'organisation et d'associations décrivant les relations entre ces objets. (la figure 3.8. en est un exemple)

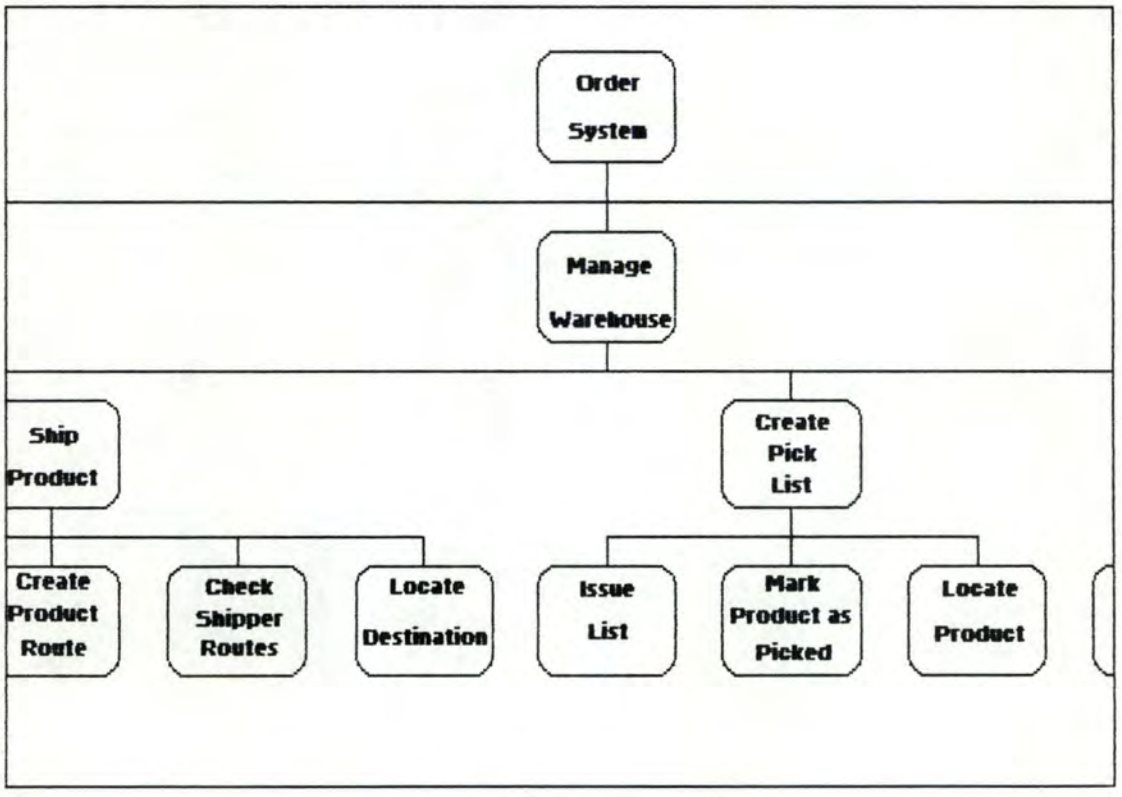


figure 3.8.

Le diagramme des flux permet de décrire le passage des données à travers les fonctions d'une activité. Un diagramme des flux est composé

- de sous-activités composant l'activité principale (qui peuvent être également décomposées en un nombre infini d'activités secondaires),
- de fichiers de données servant de dépôt pour les données produites,
- d'agents externes qui sont des composants situés à l'extérieur des limites du modèle et qui sont capables de produire et de recevoir des données,
- et de flux de données qui transportent les données entre les activités.

(La figure 3.9. est un exemple de diagramme de flux.)

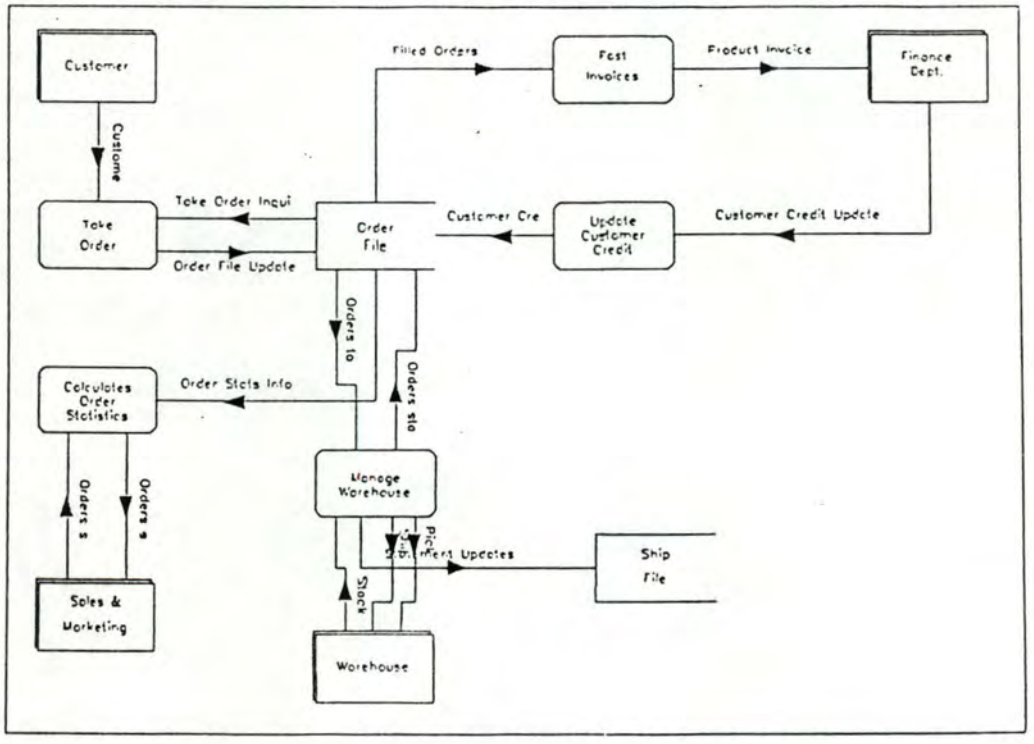


figure 3.9.

Le diagramme Entités-Relations contient les types d'entités impliqués dans l'organisation et les relations entre elles. Il est composé des types d'entités, des types de relations (représentées par des lignes entre les entités) et des contraintes quantitatives (ou connectivités). (La figure 3.10. présente un exemple de modèle Entités-Relations)

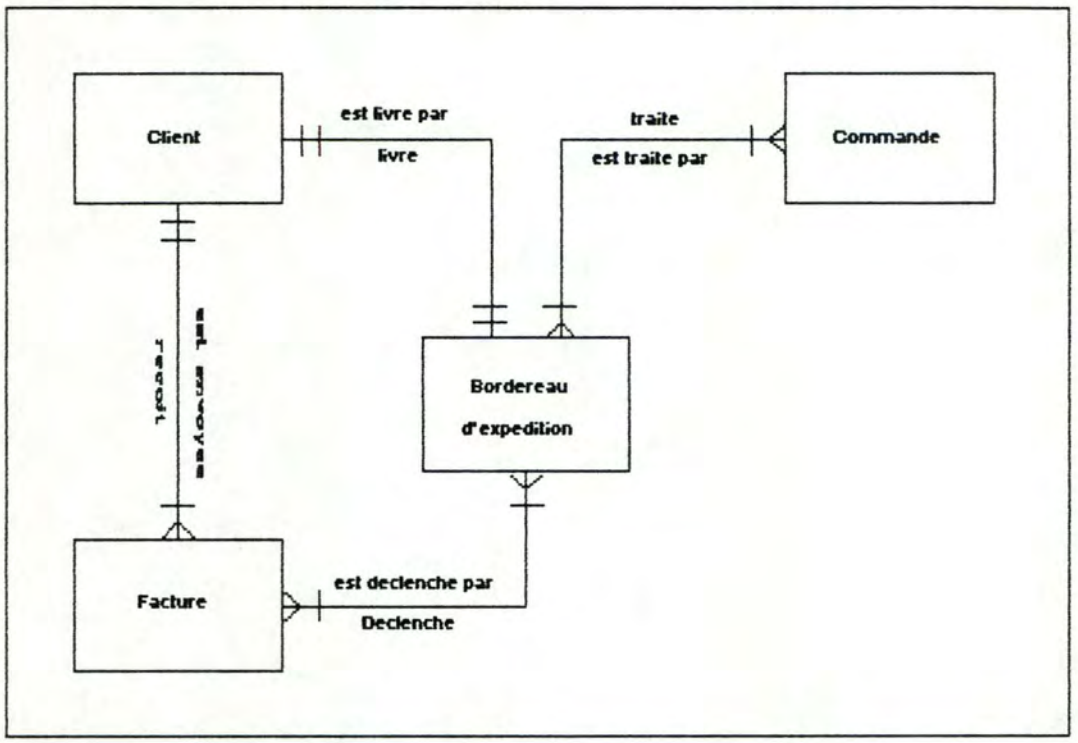


figure 3.10.

Enfin, le diagramme d'actions est un instrument graphique permettant de représenter la logique d'un programme de manière simple et facile à comprendre. Il est composé d'étapes à suivre afin d'atteindre un but. (La figure 3.11. présente un exemple de diagramme d'actions)

figure 3.11.

Ces quatre outils sont reliés entre eux de manière à pouvoir visualiser la même information de différentes façons. Les informations saisies dans chacun de ces diagrammes sont mémorisées dans l'encyclopédie. Une mise à jour d'un diagramme est donc toujours répercutée immédiatement dans l'encyclopédie, et une modification de celle-ci entraîne une mise à jour de tous les diagrammes affichés à l'écran. Une modification dans une fenêtre est donc toujours répercutée immédiatement dans toutes les autres.

3.2.2.2. Remarques concernant l'éditeur de saisie de "I.E.W."

Suite à la démonstration à laquelle nous avons assistés, nous pouvons nous permettre de formuler les remarques suivantes quant à l'utilisation de "I.E.W." :

- Du point de vue des contrôles sur la saisie (intégrité, cohérence, sémantique, complétude, ...) : ils sont effectués a posteriori par

"Knowledge Coordinator", système expert écrit en PROLOG et contenant environ 1200 règles d'inférence.

- Représentation graphique : toutes les informations de type texte sont saisies via une "popup-window" (fenêtre supplémentaire).
- Modifications : comme nous l'avons signalé au paragraphe précédent, les modifications d'un objet dans une fenêtre sont toujours répercutées dans toutes les fenêtres concernées, immédiatement.

  Les connexions peuvent être sélectionnées en se positionnant sur leur label (leur nom), les objets en se positionnant sur ceux-ci.

3.2.2.3. Remarques concernant l'éditeur de saisie et l'éditeur de restitution de "I.E.W."

Les remarques suivantes sont aussi pertinentes pour l'éditeur de saisie que pour l'éditeur de restitution :

- Représentation graphique : la représentation des noms d'objets et des noms de connexions sont tronqués en fonction de la place disponible.

  Quel que soit le diagramme utilisé, les seules lignes pouvant apparaître sont des lignes horizontales et des lignes verticales. Il n'y a jamais de connexions en diagonale.

  La représentation des objets (leur forme) est dynamique et peut être changée à n'importe que moment. Cette modification est répercutée immédiatement dans toutes les fenêtres.

  Tous les objets ont toujours la même taille l'un par rapport à l'autre. Il y a trois possibilités : soit la taille de base, c'est-à-dire une taille fixe, quelle que soit la grandeur de la fenêtre, soit la taille

"full size", proportionnelle à la grandeur de la fenêtre dans laquelle est représentée le diagramme, soit enfin une taille qui est fonction de la taille de base (15 à 200% de la taille de base).

3.3. Contribution au développement de l'éditeur graphique IDA

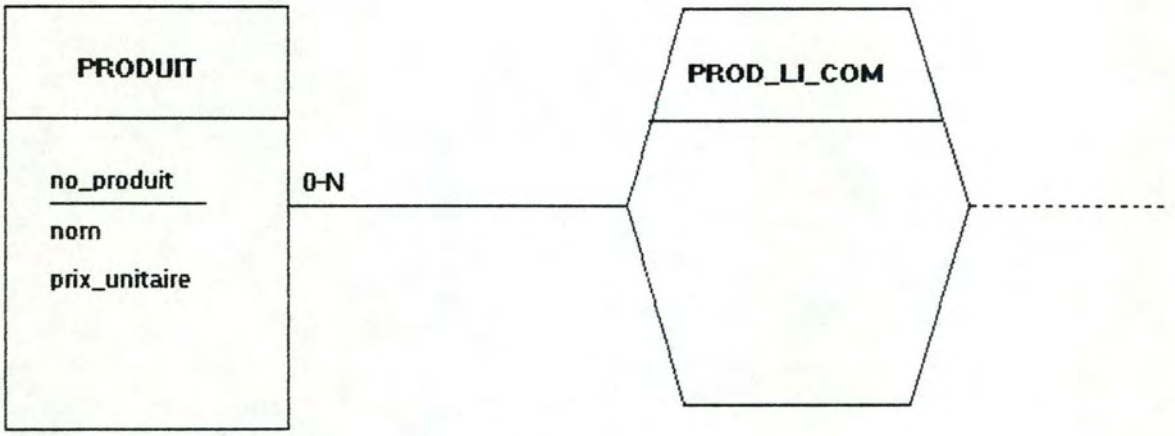
3.3.1. Le problème

Comme nous l'avons souligné dans le chapitre 3, l'utilisateur de IDA se trouve confronté à deux problèmes majeurs lors de la saisie de son analyse à l'aide d'un éditeur texte :

D'une part se pose un problème de traduction de sa vision du système d'information en phrases D.S.L., et d'autre part un problème de visualisation des modèles n'ayant pas une structure hiérarchique : aucune représentation globale graphique de tels aspects du système d'information ne sont disponibles (exemple : un modèle E.R.A., ...).

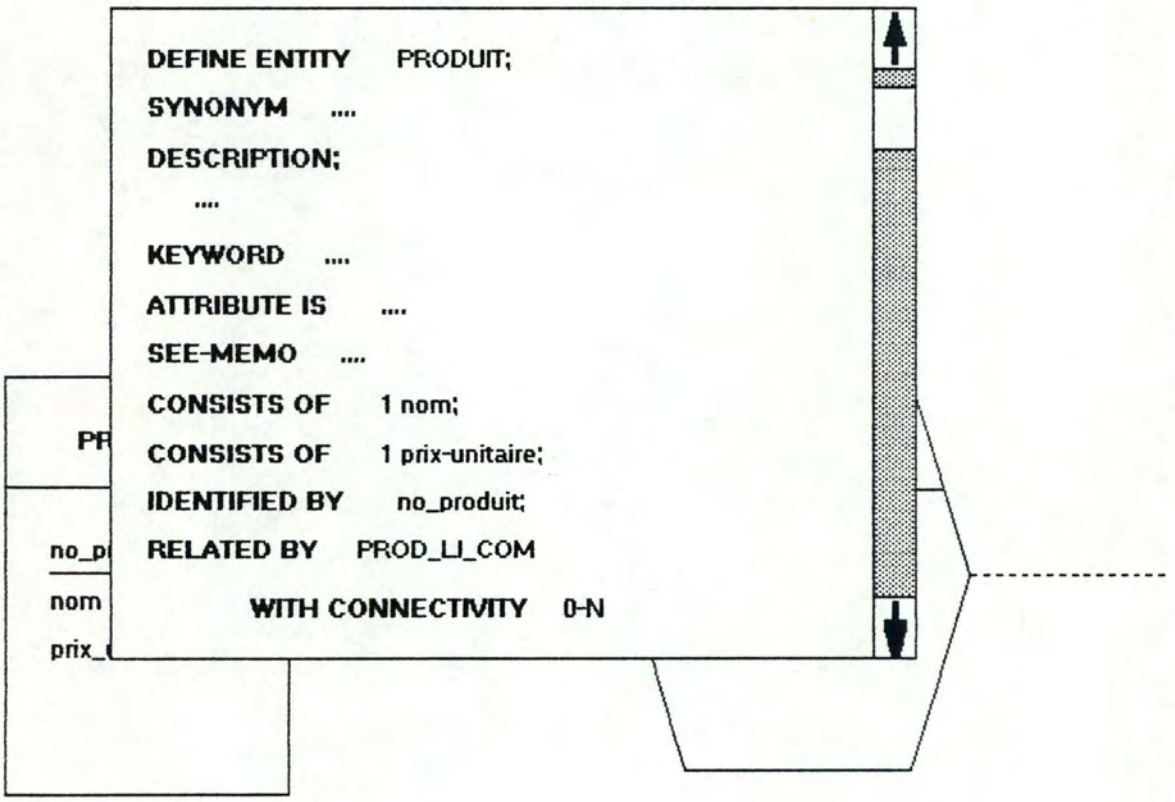
La saisie graphique de son analyse permettrait de palier ces inconvénients. Néanmoins, une réflexion sur l'étendue de cet éditeur s'impose. En effet, le but recherché est d'améliorer la clarté de la base de données en en donnant une vue globale. Vouloir tout représenter graphiquement aboutirait à un résultat contraire. On peut difficilement imaginer une représentation graphique d'une entité, en ce compris ses synonymes, sa description, les processus qui l'utilisent, ses attributs ou groupes d'attributs, ses identifiants, ... sur un seul et même dessin.

Il est donc clair qu'un compromis s'impose entre le texte et les graphiques. Pour l'entité par exemple, on pourrait représenter graphiquement son nom, ses identifiants, ses attributs et ses connectivités avec les associations auxquelles elle est reliée (figure 3.12.a). Un "zooming" sur l'entité permettrait de revenir à un éditeur texte et de compléter sa description (figure 3.12.b). Puisque l'éditeur texte existant a été développé sous MS-WINDOWS, nous pourrions éventuellement l'adapter pour l'utiliser dans ce cadre. Les modifications à apporter seraient beaucoup moins coûteuses que la conception d'un nouvel éditeur.



Représentation graphique d'une entité

figure 3.12.a



Zooming texte sur cette entité

figure 3.12.b

3.3.2. Proposition concernant l'utilisation de l'éditeur graphique

Dans notre étude, nous avons considéré deux possibilités de mise en page pour un éditeur graphique.

La première (figure 3.13.) comporte trois fenêtres :

- la page graphique dans laquelle l'utilisateur dessine son diagramme
- la zone des menus contenant l'ensemble des options disponibles, en fonction de ce que fait l'utilisateur et en fonction de sa situation dans l'éditeur
- la zone des icônes dans laquelle sont représentées les formes correspondant à chaque objet graphique.

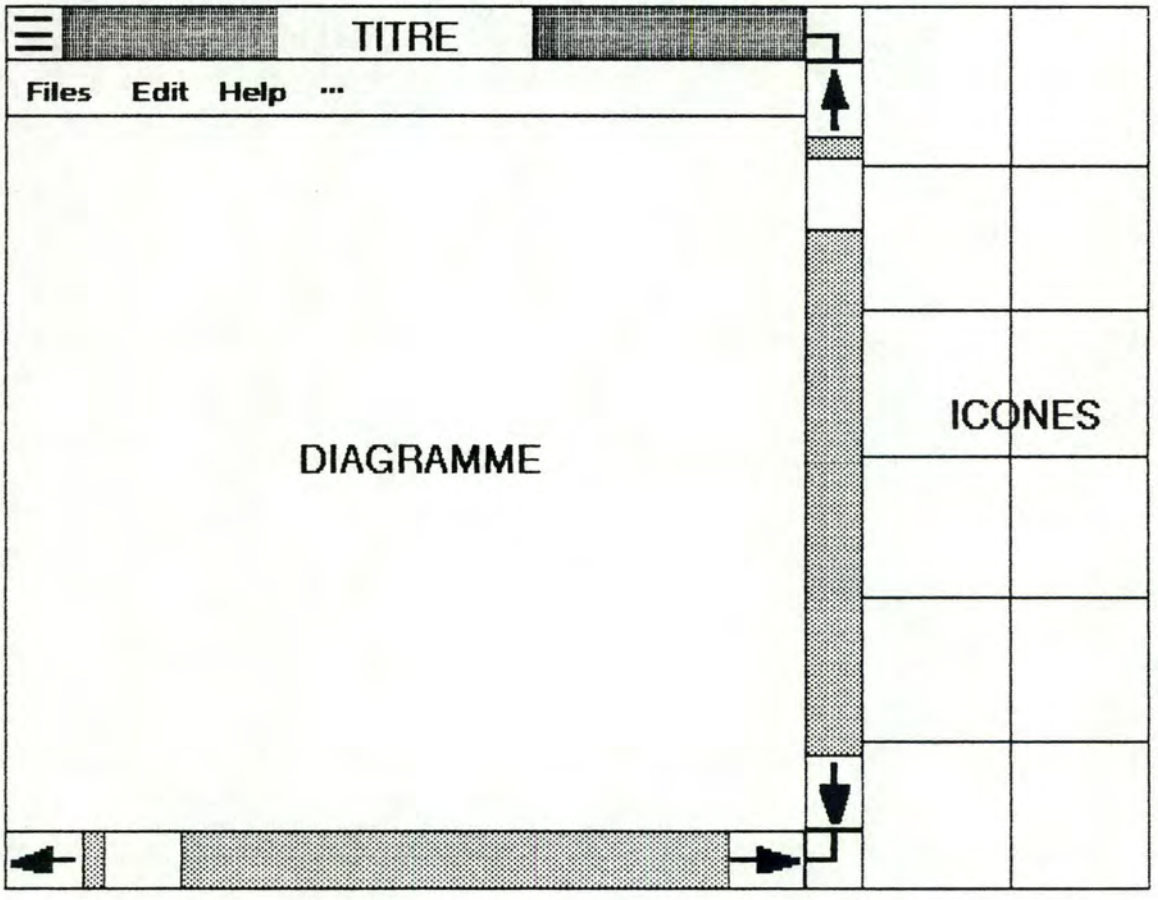


figure 3.13.

La seconde (figure 3.14.) ne comporte que deux fenêtres : la page graphique et la zone des menus. L'utilisateur peut accéder aux objets via une des options des menus. Si cette méthode semble moins agréable de prime abord, elle a le grand avantage d'offrir une page graphique plus grande.

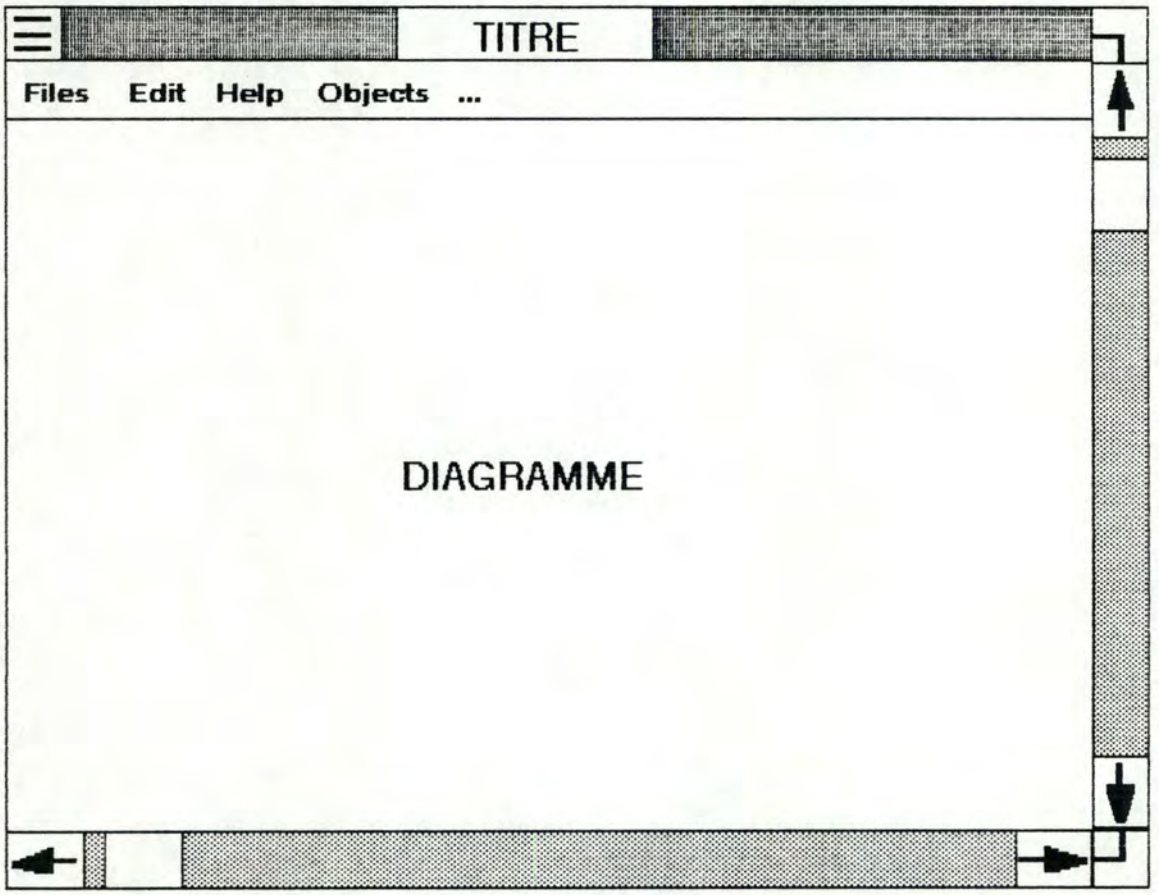


figure 3.14.

Pour chacune de ces deux possibilités, des fenêtres contenant du texte peuvent venir se greffer. Elles permettront la saisie de tous les éléments du langage n'ayant pas de représentation graphique.

3.3.3. Architecture fonctionnelle d'un système de saisie

3.3.3.1. Vue générale

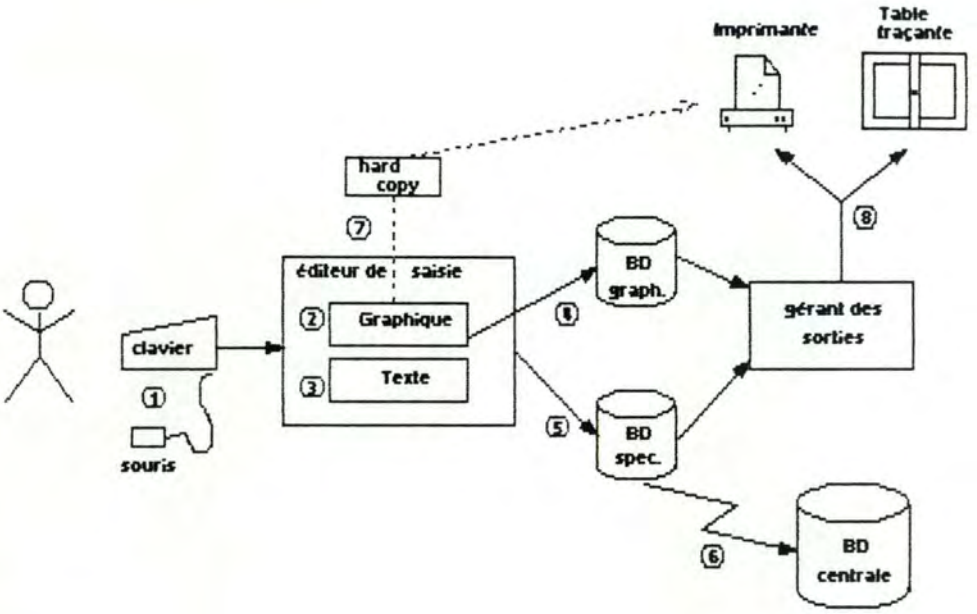


figure 3.15.

L'utilisateur est en communication avec l'éditeur de saisie grâce d'une part à son clavier, et d'autre part à une souris (1). Les deux périphériques de saisie lui permettent de créer son diagramme, soit de façon graphique (2) (si une représentation graphique existe), soit par l'intermédiaire d'un éditeur texte (3). La saisie graphique se fait par une manipulation d'objets à l'écran, la saisie texte en complétant des "templates".

L'éditeur de saisie gère deux bases de données locales, l'une graphique qui est en relation directe avec la partie graphique de l'éditeur (4), l'autre qui contient les spécifications (aussi bien en relation avec la partie texte qu'avec la partie graphique de l'éditeur).

A la fin de l'édition, la base de données des spécifications est envoyée sur le "main frame" via un réseau (6), pour mettre à jour la base de données centrale.

Si l'utilisateur le désire, il peur effectuer une impression de son modèle, soit en demandant une "hard-copy" de son écran (7), soit en lançant une commande d'impression qui reproduit tout le schéma depuis les bases de données (8).

3.3.3.2. L'éditeur de saisie

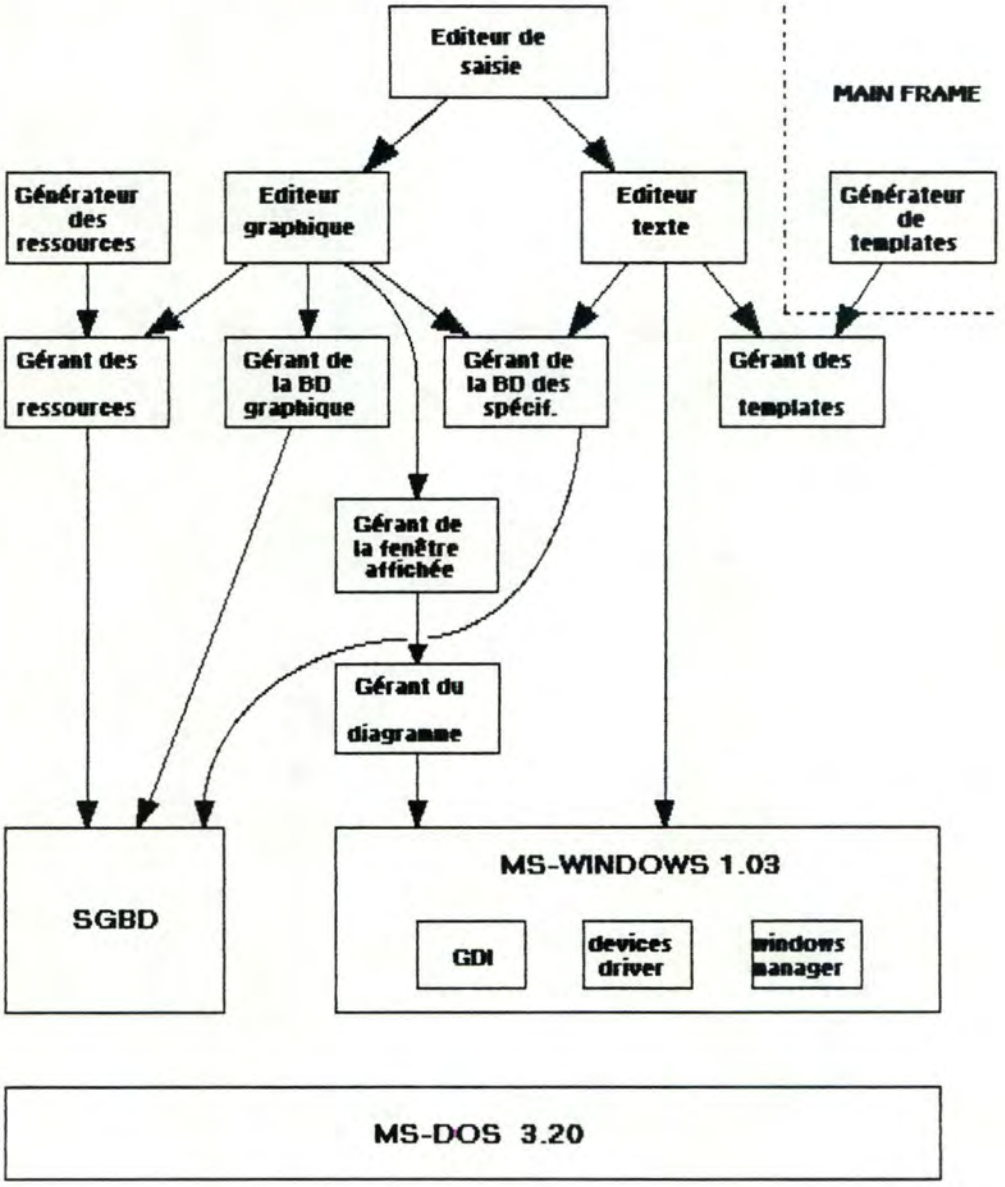


figure 3.16.

niveau 0 : O.S. : Le système d'exploitation MS-DOS 3.20

niveau 1 : Le système de gestion de base de données.

MS-WINDOWS 1.02 ou 1.03 : logiciel de gestion de fenêtres (bureau électronique), de menus, ..., logiciel de gestion graphique (GDI : Graphics Display Interface), gérant des périphériques (imprimantes, tables traçantes, cartes graphiques, souris, ...)

niveau 2 : Le gérant du diagramme : ce module a pour fonction la gestion du diagramme dans le monde graphique. Le monde graphique est un espace de travail indépendant du mode de résolution et des spécificités de l'espace physique.

niveau 3 : Le gérant de la fenêtre affichée : ce module a pour fonction le zooming et la gestion de la fenêtre dans le monde graphique.

niveau 4 : Le gérant des ressources : ce module doit assurer la gestion de la représentation graphique, à savoir, la représentation des objets graphiques configurée par l'utilisateur (il choisit ce qu'il veut représenter et comment il veut le représenter).

Le gérant des templates : ce module a la même fonction que le module précédent, mais au niveau du texte : il gère les formes à remplir par l'utilisateur en fonction de la syntaxe du langage.

Le gérant des entrées / sorties sur bases de données : il est composé de deux modules, à savoir la mémorisation des graphiques et la mémorisation des spécifications au niveau local

niveau 5 : Le générateur des ressources : c'est une interface qui permet à l'utilisateur de spécifier, de façon simple et agréable la liste des

objets qu'il veut voir apparaître en mode graphique et leur représentation à l'écran.

Le générateur de templates : ce module traduit automatiquement la "méta" en templates. Ce module peut être implémenté sur le "main frame". (Il ne servira que lors de modifications dans la structure du langage)

L'éditeur graphique et l'éditeur de texte : ce sont deux composants de l'éditeur de saisie. L'éditeur graphique permet la saisie des objets sous forme de dessins et l'éditeur texte assure le remplissage des templates.

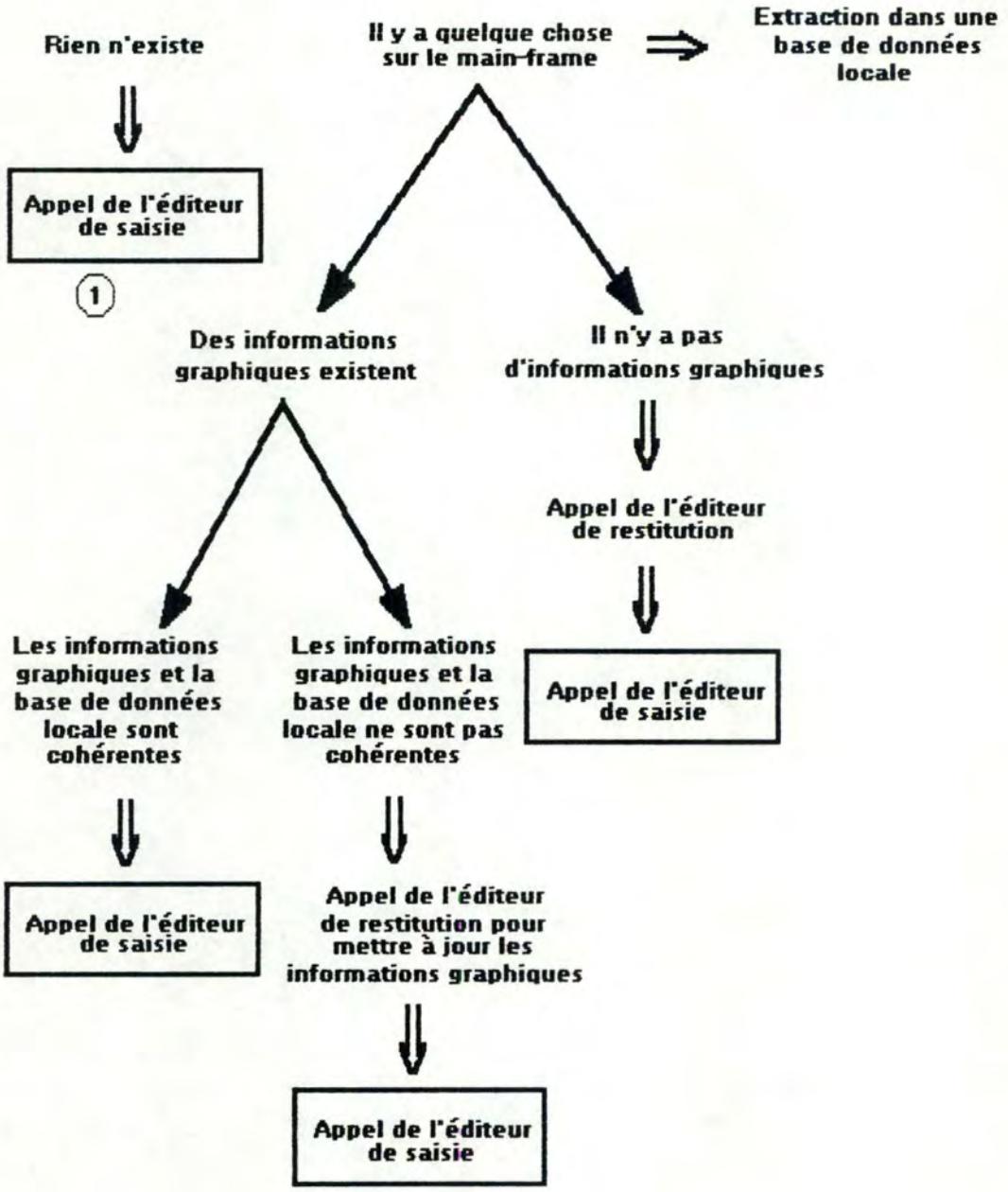
niveau 6 : L'éditeur de saisie : c'est le pilote pour la saisie. Il est également responsable des contrôles syntaxiques et sémantiques aussi bien au niveau du traitement du texte qu'au niveau du traitement des graphiques.

3.3.4. Proposition d'un scénario

Si on considère l'éditeur de saisie comme un un support d'expression qui doit guider l'analyste à saisir toutes les informations qu'il a relevées, lorsque celui-ci désirera utiliser la station de travail graphique, il ne le fera qu'après avoir terminé l'analyse de l'aspect qu'il désire encoder.

Il devra alors respecter les étapes suivantes :

* L'utilisateur choisit l'aspect D.S.L. sur lequel il souhaite travailler.
* Il choisit ensuite les objets (occurrence) avec lesquels il désire travailler.
* Le système effectue alors un contrôle d'existence :



Il est à noter que lorsque l'éditeur de saisie est appelé, les deux bases de données locales sont cohérentes. Nous allons aborder ces problèmes de cohérence dans le paragraphe suivant. Remarquons également que sauf dans le cas (1), la représentation des objets existants est affichée à l'écran.

Une fois face à son graphique, l'utilisateur peut le modifier ou le compléter. Pour chaque modification effectuée, le programme d'application effectue un maximum de contrôles immédiats : syntaxe, cohérence, sémantique, ...

Pour passer d'un modèle à l'autre, l'utilisateur a la possibilité d'ouvrir une seconde fenêtre. S'il a des objets à récupérer dans un modèle pour les introduire dans un autre, il a la possibilité de les sélectionner, et par un simple clic de la souris, de les insérer dans son nouveau modèle. (méthode "cut and paste")

Le sauvetage de sa base de données locale des spécifications dans la base de données centrale se fera uniquement à sa demande.

3.3.5. Problème de cohérence entre les bases de données

Le problème de cohérence entre les bases de données se situe à deux niveaux (figure 3.17.).

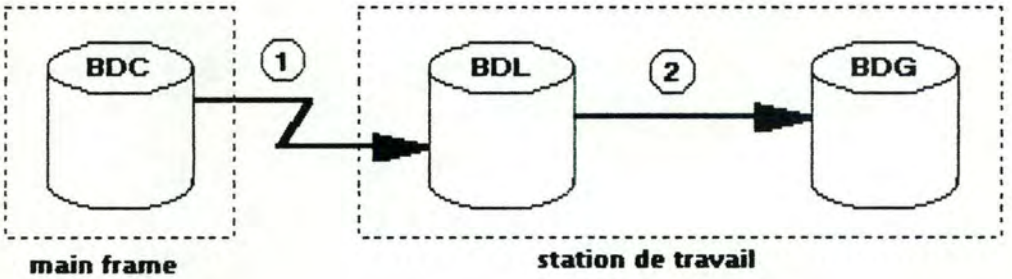


figure 3.17.

Le problème de la cohérence entre la base de données centrale et la base de données locale est dû au fait que la base de données centrale est multi-utilisateurs. C'est donc un problème de concurrence (figure 3.18.).

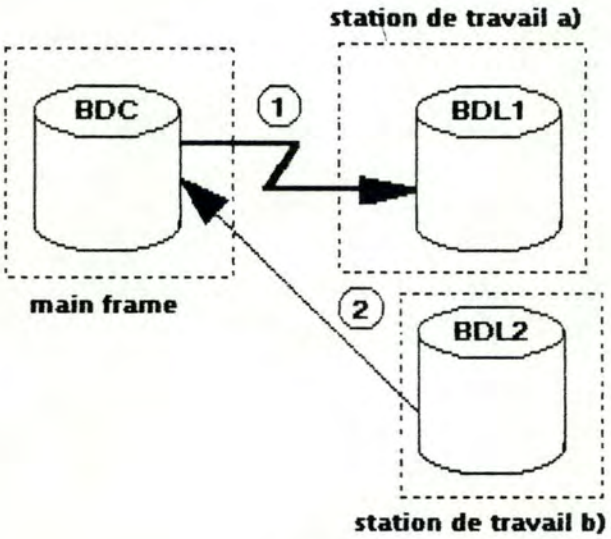


figure 3.18.

(1) L'utilisateur de la base de données locale 1 extrait sa base de données de la base de données centrale.

(2) Après cette extraction, l'utilisateur de la base de données locale 2 met à jour la base de données centrale à partir de sa base de données.

La base de données locale 1 peut donc ne plus correspondre à ce qui est dans la base de données centrale.

Le deuxième problème est la cohérence entre la base de données locale et la base de données graphique.

Considérons la situation générale suivante :

Un utilisateur vient de terminer sa session de travail, sa base de données locale et sa base de données graphique sont cohérentes. Au début de la session suivante, il désirera s'assurer que sa base de données locale correspond toujours à la base de données centrale. Il extraira une nouvelle base de données locale avec exactement les mêmes

critères que la fois précédente. Il est nécessaire de comparer les deux bases de données locales (l'ancienne et celle qu'il vient d'extraire) et de détecter les différences de manière à mettre à jour la base de données graphique (via l'éditeur de restitution).

Trois cas peuvent se présenter :

- Un objet a été supprimé dans la base de données des spécifications : on doit le supprimer de la base de données graphique, éventuellement avec tout ce qui s'y rapporte. (Suite à la confirmation de l'utilisateur.)
- Un objet a été ajouté dans la base de données des spécifications : on propose à l'utilisateur de l'ajouter à la base de données graphique.
- Un objet a été modifié dans la base de données des spécifications:

* soit la modification n'a aucun impact sur le graphique.

exemple :

```
DEFINE ENTITY employé;
   CONSIST OF nombre prénom-employé;
   ...
DEFINE SYSTEM-PARAMETER nombre;
   VALUE IS 3;
```

qui devient

```
DEFINE SYSTEM-PARAMETER nombre;
   VALUE IS 5;
```

Si l'utilisateur décide de ne pas représenter le "SYSTEM-PARAMETER name" graphiquement dans le schéma E.R.A., la modification n'affecte que la base de données locale, donc la base de données graphique reste inchangée.

* soit la modification a un impact sur le graphique.

exemple :

```
DEFINE PROCESS A;
   ON TERMINATION TRIGGERS B;
DEFINE PROCESS B;
   ON TERMINATION TRIGGERS C;
```

qui devient

```
DEFINE PROCESS A;
   ON TERMINATION TRIGGERS B;
   ON TERMINATION TRIGGERS C;
```

La modification est directement reportée sur le graphique avec interaction de l'utilisateur.

Pour gérer la cohérence entre la base de données locale et la base de données graphique, nous avons étudié deux méthodes.

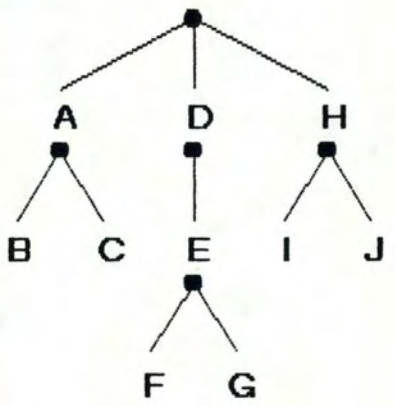
1. Les listes structurées

   Cette méthode consiste à représenter chaque base de données par une liste structurée et de les comparer. Le problème n'est pas aussi simple qu'il ne semble l'être, deux listes pouvant être sémantiquement identiques et physiquement différentes.

Exemple : A
. B
. C
D
. E
. . F
. . G
H
. I
. J

A
. C
. B
H
. I
. J
D
. E
. . G
. . F

Afin d'éviter ce problème, nous proposons pour la liste la représentation suivante :



Nous aurons une représentation arborescente. Au sein de chaque noeud, les éléments seront triés arbitrairement, par exemple par ordre lexicographique. Grâce à ce tri, les listes pourront être comparées niveau par niveau (il s'agit d'une simple comparaison d'arbres pour laquelle il existe des algorithmes).

Si cette méthode a l'avantage de détecter facilement les différences entre les bases de données, elle possède néanmoins un ensemble d'inconvénients. La création et l'ordonnancement des listes

structurées peut se faire aisément à partir de la base de données locale, mais ce travail est beaucoup plus difficile à partir de la base de données graphique.

Nous proposons donc une gestion de la liste structurée associée à la base de données graphique en temps réel, c'est-à-dire chaque fois que l'utilisateur modifie le graphique, la modification se répercute sur la liste triée. La liste correspondant à la base de données graphique est donc continuellement à jour.

2. La base de données graphique est en relation constante avec sa base de données locale (éventuellement, une seule base de données avec deux sous-schémas (figure 3.19.)).

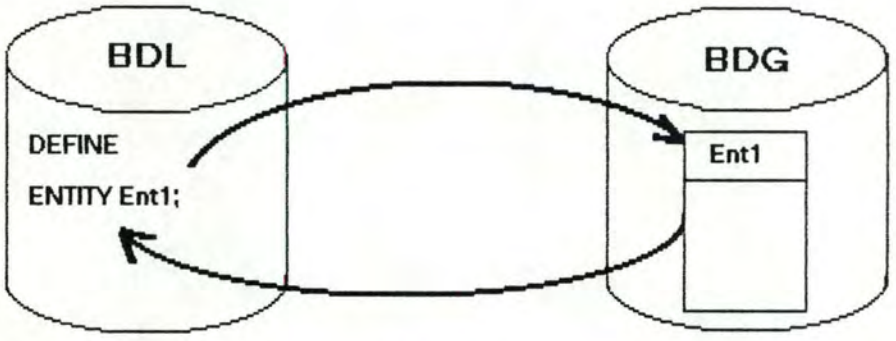


figure 3.19.

Chaque élément de la base de données locale est associé à l'élément correspondant de la base de données graphique et vice et versa.

La gestion de la cohérence consiste en une "simple" comparaison entre la base de données locale et l'extraction qui vient d'être faite à partir de la base de données centrale.

Tout comme la première méthode, celle ci possède des avantages et des inconvénients. L'avantage le plus important semble être la possibilité de comparer directement les objets de la base de données graphique et les objets de la base des données des spécifications. Les objets à ajouter à la base de données graphique sont très faciles à détecter, ceux à supprimer le sont moins, un parcours séquentiel de la base de données des spécifications étant nécessaire en fin de comparaison pour relever les objets qui n'ont pas de correspondant graphique. Le problème le plus important est la détection de la modification d'un objet. En effet, une simple comparaison au niveau des objets n'est pas suffisante. Si un objet a été modifié, une comparaison "en profondeur" (au niveau des relations) s'avérera nécessaire pour le détecter. Il s'agit là d'un problème énorme que nous ne pouvons pas aborder dans le cadre de ce mémoire.

Pour comparer les deux bases de données, nous proposons de marquer les objets consultés et ajoutés. Les objets non marqués sont à supprimer.

Présentation d'un exemple : Soit le contenu de la base de données locale : entity-1, entity-2, entity-3 et le contenu de la base de données extraite : entity-1, entity-3, entity-4. Le parcours séquentiel de la base de données extraite permet de reconnaître l'existence dans la base de données locale des objets entity-1, entity-3. Un indicateur sera positionné pour chacun de ces objets. Pour l'objet entity-4, l'ajout sera effectué dans la base de données locale, l'indicateur positionné, et il sera demandé à l'utilisateur s'il désire représenter cet objet graphiquement. Dans la négative, il sera marqué différemment pour

indiquer qu'il n'appartient pas à la base de données graphique. Après cette opération, un parcours séquentiel de la base de données locale permettra de reconnaître les objets à supprimer, c'est-à-dire ceux qui n'auront pas été marqués.

Les modifications pourraient être gérées comme des suppressions suivies d'ajouts (suivant l'importance de la modification à effectuer). Cette méthode est très simple à gérer mais peut ne pas être conviviale puisqu'elle risque d'obliger l'utilisateur à recréer plusieurs fois une même partie de son schéma.

3.4. Proposition d'un système ouvert : le SEEOF

3.4.1. Présentation du SEEOF

Le SEEOF (Software Engineering Environment Of the Future) est un logiciel en cours de développement à l'université du Michigan (U.S.A.). Il s'agit d'un environnement de travail destiné à faciliter la tâche aux analystes et aux programmeurs dans la conception de logiciels [PRIS86a].

Le but principal du SEEOF est d'offrir un environnement de développement de logiciels évolué qui permettrait aux programmeurs ou aux groupes de programmeurs de produire un système d'information rapidement et meilleur marché en accroissant la productivité et la qualité de l'output. Cette qualité de l'output peut être atteinte en utilisant des techniques d'interaction avancées pour rendre plus accessible à l'utilisateur toutes les possibilités du système et en utilisant des techniques d'intelligence artificielle et des modèles de base de données plus élaborés qui automatiseraient la production de code et la vérification de programmes.

La particularité fondamentale du SEEOF est son indépendance vis-à-vis de toute méthodologie. Cet environnement se veut très général et apte à fonctionner avec n'importe quelle méthode d'analyse, toute la méthodologie étant connue via une base de connaissance.

Le SEEOF est constitué d'un ensemble de logiciels appelés *Serveurs*. Chaque serveur peut communiquer avec un autre via une interface bien définie. (Un langage de commandes.) Cette architecture s'applique facilement aux systèmes répartis. (figure 3.20.)

Le composant du SEEOF responsable de toutes les interactions avec l'utilisateur est le **DES** (Display Edit Server). C'est la partie du logiciel sur laquelle nous avons travaillé.

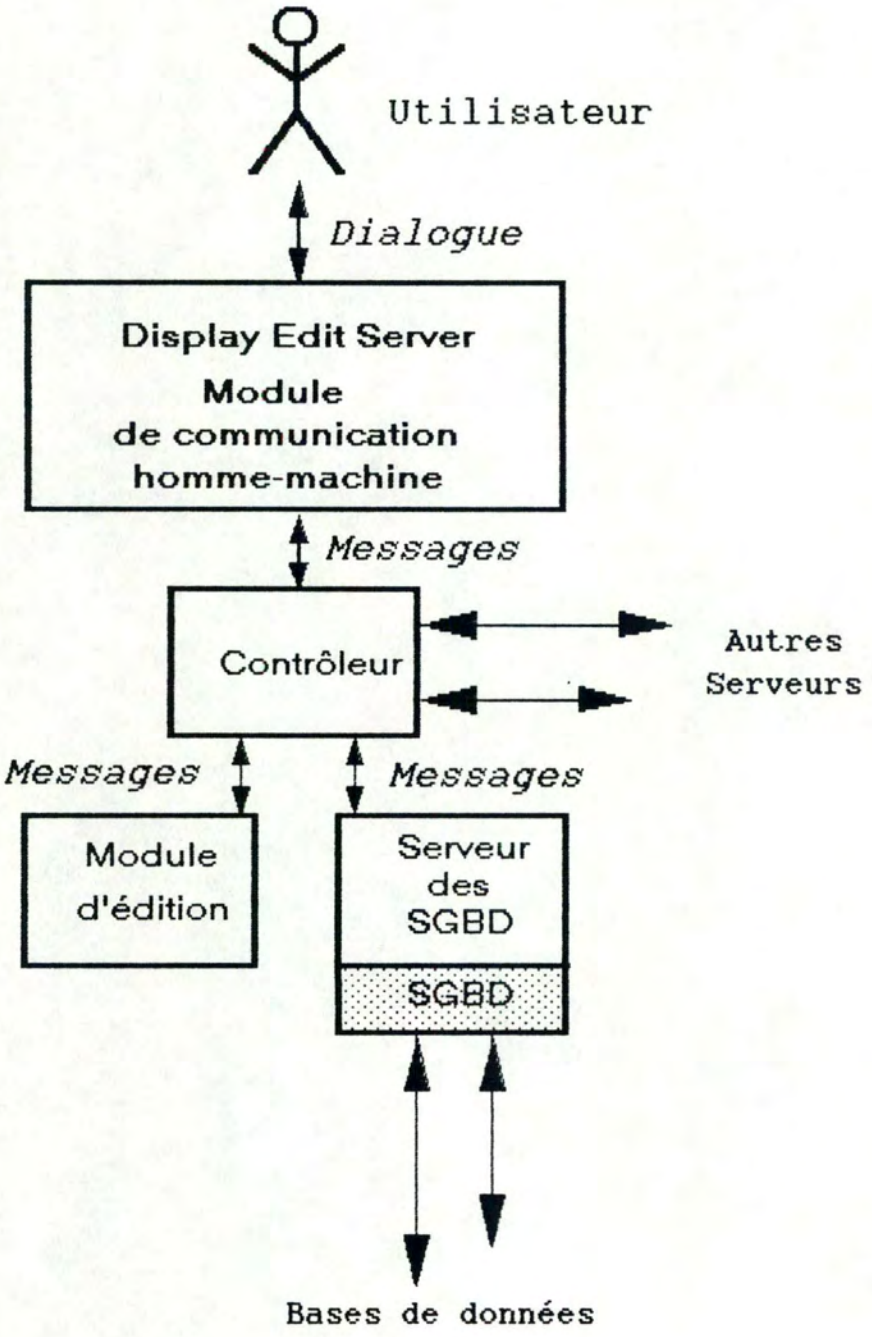

Utilisateur
Dialogue
Display Edit Server
Module
de communication
homme-machine
Messages
Contrôleur
Autres
Serveurs
Messages
Messages
Module
d'édition
Serveur
des
SGBD
SGBD
Bases de données

figure 3.20.

3.4.2. L'interface homme-machine : le DES

Dans un sens, le DES est un "package" de traitement graphique interactif de très haut niveau. Le DES utilise des fenêtres, des icônes, des "pull-down" menus, une souris ou une table à digitaliser et n'importe quel autre équipement pouvant rendre l'interface utilisateur plus intuitive et plus naturelle. Du point de vue de l'utilisateur, le DES est un outil d'édition générale. En utilisant le DES, l'utilisateur est capable d'afficher et d'éditer tous les aspects du système cible qui auront été définis dans la méthodologie (le système cible est une occurrence du système d'information qui est le but d'un projet de développement particulier). L'utilisateur peut aussi exécuter différentes fonctions en parallèle en utilisant plusieurs invocations du DES. Chacune de ces occurrences apparaît dans sa propre fenêtre (système multi-fenêtres).

Le DES est en relation avec le monde extérieur via quatre interfaces externes:

3.4.2.1. L'interface utilisateur

C'est la vue du logiciel et de ses fonctionnalités par l'utilisateur. La littérature classe généralement ces interfaces en quatre catégories [PRIS86b] : les langages de commandes, la communication par menus, le "bureau électronique" (environnements multi-fenêtres de type MACINTOSH ou MS-WINDOWS) et les interfaces du futur (reconnaissance de la voix, de la pensée, de l'écriture manuscrite, ...)

Le DES est tout d'abord une interface de type "bureau électronique", mais il est également capable de fonctionner avec un langage de commandes ou de lancer des applications en "batch".

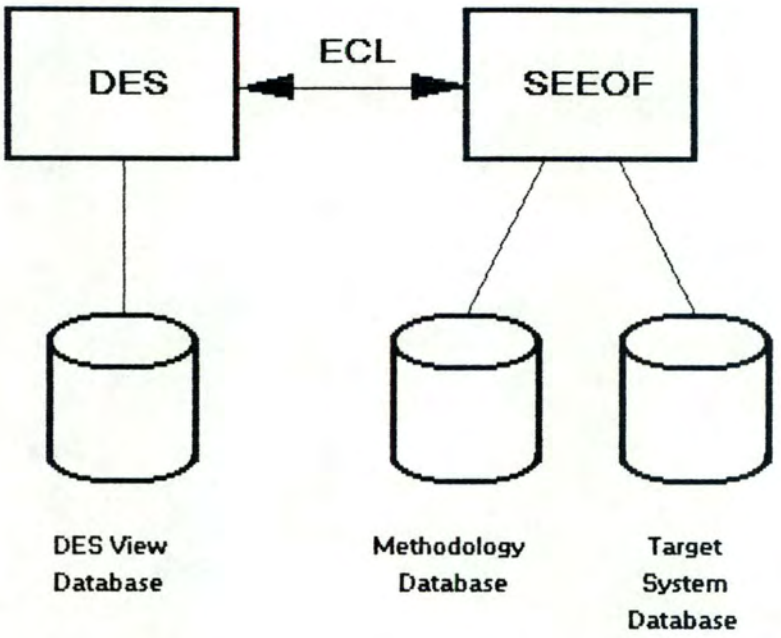
3.4.2.2. L'interface hardware

Le SEEOF est un logiciel qui doit être implémentable sur différents ordinateurs. L'interface hardware est l'interface entre le logiciel et la machine. L'interface hardware du DES est le système d'exploitation de la machine sur laquelle il est implémenté. En effet, étant donné le manque de standard pour les environnements de type "bureau électronique", une occurrence du DES devra être implémentée pour chaque machine. Il faut néanmoins noter que dépendance vis-à-vis d'une machine n'est pas synonyme de dépendance vis-à-vis des périphériques, c'est-à-dire qu'une instanciation du DES à une machine donnée doit pouvoir fonctionner par exemple avec n'importe quelle imprimante simplement en ajustant quelques paramètres.

3.4.2.3. L'interface software

Le DES interagit avec le SEEOF en échangeant des messages (types abstraits de données) avec le "control server".

Le flux de messages présenté sur la figure 3.21. est l'interface software du DES. Le type abstrait de données est appelé vue et les opérations disponibles pour manipuler ces données sont collectionnées dans "l'Edit Control Language" (ECL).



Maintenance of View definitions by the DES

figure 3.21.

3.4.2.4. L'interface "méthodologie"

Comme nous l'avons vu auparavant, le SEEOF se veut indépendant de toute méthodologie. L'occurrence de la méthodologie que l'utilisateur désire employer doit donc être décrite. Cette description se constitue de deux parties. La base de connaissances contient les informations nécessaires pour spécifier et analyser le système cible sous une méthodologie donnée. La partie graphique contient les informations nécessaires pour afficher et éditer les différentes vues du système cible sous une méthodologie donnée.

3.4.3. Architecture d'une instanciation du DES au langage D.S.L.

Lors de notre stage à l'université du Michigan, nous avons développé une instanciation du DES au langage D.S.L., c'est-à-dire une instanciation de la base de connaissances et de la partie graphique à D.S.L. La figure 3.22. en présente l'architecture que nous allons détailler.

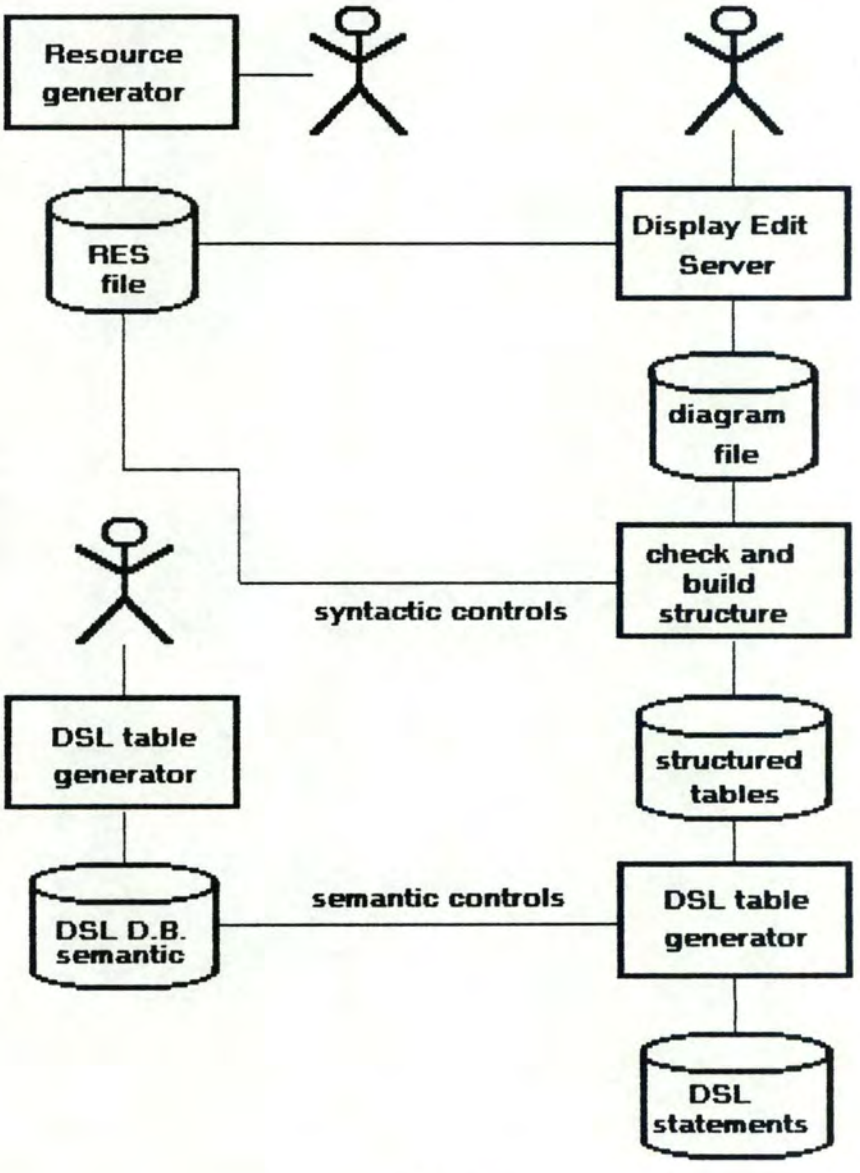


figure 3.22.

L'instanciation du DES à une méthodologie fixée (par exemple D.S.L. ou un des aspects de D.S.L.) se fait à deux niveaux.

Le premier comprend la création du fichier des ressources (RES file) qui contient les caractéristiques graphiques et syntaxiques pour chaque objet et chaque connexion. Ce fichier peut être créé via un générateur (sorte de compilateur) permettant de spécifier ces caractéristiques dans un langage évolué. Dans l'instanciation que nous avons développée, chaque objet est décrit en utilisant la méthode "turtle-graphics" : tous les objets sont dessinés par rapport à une case de taille fixe (100 x 100 points dans l'exemple) en déplaçant une plume de façon à tracer des lignes. (Remarque : le concept de "plume" provient de l'utilisation de la table traçante : un déplacement "plume haute" correspond à un mouvement du bras de la table traçante avec la plume qui n'est pas en contact avec le papier. Il s'agit donc d'un positionnement à une coordonnée donnée. Un déplacement "plume basse" correspond au tracé d'une droite, partant de la position courante vers la position spécifiée comme paramètre.)

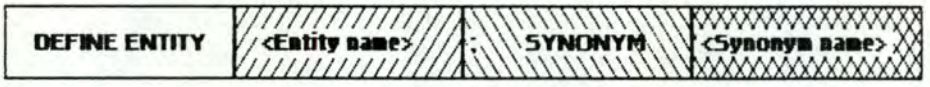
C'est également dans ce fichier que sont définis les attributs des objets graphiques et des connexions.

exemple : (extrait du fichier des ressources)

```
[OBJECTS]          (=> ce qui suit est la définition des objets)
*
*                  ( ligne de commentaire )
*
Entity             ( 1ère ligne = nom de l'objet)
0     0    2       ( 1er nombre : coord. en X)
100   100  2       ( 2ème nombre : coord. en Y)
0     30   0       ( 3ème nombre : type de trait :
100   30   1         0 = déplacement plume haute
1     2    10        1 = déplacement plume basse
99    27   10        2 = rectangle
                     10 = position du nom de l'objet )
```

Le deuxième niveau comprend la création de la base de données qui contient les phrases D.S.L. et leur sémantique (sous forme de "templates"). Cette base de données peut également être créée via un générateur à partir d'un langage de haut niveau.

Exemple de template :



Zone à remplir

Zone optionnelle

figure 3.23.

Après l'instanciation, l'utilisateur peut créer son diagramme avec le DES en utilisant les objets et les connexions définies dans le fichier des ressources. A ce niveau, l'utilisateur est libre de dessiner ce qu'il veut, car la sémantique de la méthodologie utilisée n'est pas encore connue. Il n'a donc aucun pilote pour l'aider dans sa tâche. Son diagramme est sauvé dans un fichier : diagram file (objets et connexions avec leurs caractéristiques et leur localisation).

Une fois son diagramme réalisé, l'utilisateur peut invoquer une demande de traduction. (Le principe est le même que celui d'un compilateur : dans un premier temps, il y a encodage du programme à l'aide d'un éditeur de texte sans aucun contrôle, et dans une deuxième étape, la compilation est effectuée) La traduction se fait donc en deux étapes.

Premièrement, il est nécessaire de créer des tables structurées à partir du diagramme et du fichier des ressources, avec contrôles syntaxiques (mise en correspondance des objets et de leurs connexions, vérification de l'existence d'un objet à l'origine et à la destination de chaque connexion, ..).

La structure de ces tables pourrait être une structure de multi-listes. (figure 3.24.) A partir de n'importe quel objet / connexion, il est facile de retrouver tou(te)s les connexions / objets associé(e)s.

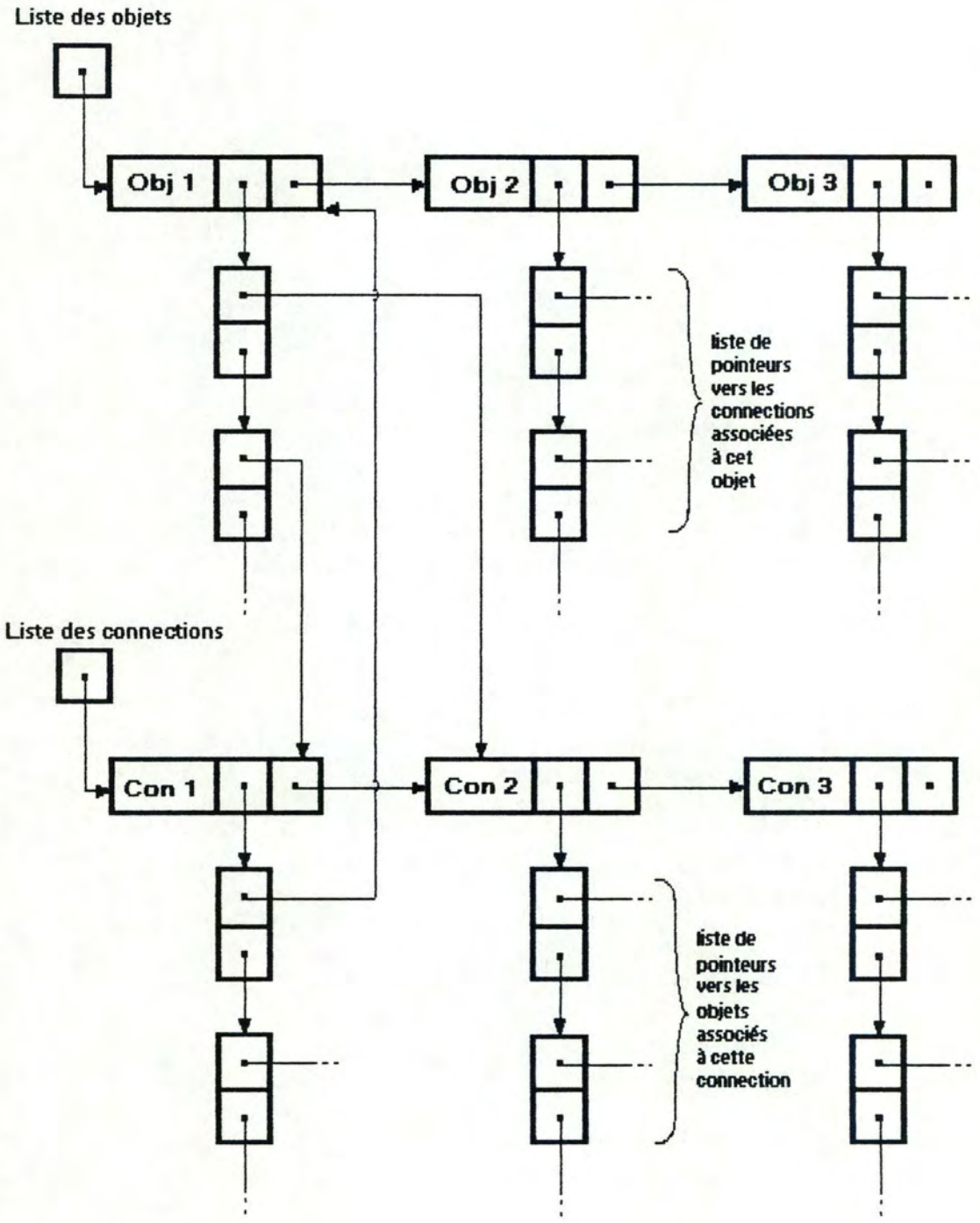


figure 3.24.

Ensuite, à partir des tables structurées et de la base de données D.S.L. (base de connaissances), le système génère les phrases D.S.L. et effectue les

contrôles sémantiques. Ce module doit être développé suivant les techniques utilisées par les compilateurs.

3.5. Conclusion

"Structured Architect" et "I.E.W." sont deux logiciels que nous avons étudiés afin de mieux comprendre les fonctionnalités d'un éditeur de saisie graphique.

"Structured Architect" permet une saisie graphique de "Structured Analysis" et son seul rôle est de traduire automatiquement un diagramme en PSL.

"I.E.W." permet une vue hiérarchisée de l'organisation étudiée (diagramme de décomposition), une description du flux des données à travers les fonctions d'une activité (diagramme des flux), la représentation des types d'entité impliqués dans l'organisation et les relations existant entre eux (diagramme Entités-Relations) et enfin la représentation de la logique d'un programme (diagramme d'actions).

L'éditeur graphique dédié à IDA est basé sur le langage de spécification D.S.L. Son but est de clarifier la base de données des spécifications en en donnant une vue globale. Vu la quantité d'information que contiennent les différents aspects du langage D.S.L., un compromis entre le texte et les graphiques s'est avéré indispensable. Cet éditeur de saisie graphique est figé puisqu'il est dépendant d'une méthodologie (D.S.L.) et donc uniquement intégrable dans IDA. Dans ce cas, il est possible de piloter l'utilisateur dans sa conception et d'effectuer un maximum de contrôles syntaxiques et sémantiques au fur et à mesure de la composition d'un diagramme.

Lors de notre analyse d'un système de saisie dédié à IDA, nous avons été confrontés à des problèmes de cohérence entre les différentes bases de données du logiciel (les bases de données centrale, locale et graphique). Nous avons présenté deux solutions pour tenter de maintenir les bases de données locale et graphique cohérentes. La première était de représenter le contenu des bases de données sous forme de listes structurées et de les comparer afin de détecter les incohérences. Mais des problèmes d'ordonnancement de listes se sont posés. Nous avons alors pensé mettre les deux bases de données en relation étroite (et éventuellement les confondre). La

comparaison paraissait alors beaucoup plus facile au premier abord. Mais si un objet a été uniquement modifié, une comparaison en "profondeur" est indispensable pour détecter cette modification.

Enfin, le "DES" est un outil d'édition générale. Il n'est lié à aucune méthodologie et est donc intégrable dans n'importe quel environnement. Mais cela implique qu'un pilotage est inconcevable et qu'aucun contrôle d'aucune sorte ne peut s'effectuer lors de la saisie. Il comprend donc une étape supplémentaire : l'interprétation du diagramme après son édition (sur demande de l'utilisateur). Cette étape revient en fait à compiler le diagramme pour le traduire suivant la méthodologie utilisée.

Dans le chapitre suivant, nous effectuerons la même analyse que dans celui-ci, mais cette fois en ce qui concerne les éditeurs de restitution graphique.

Chapitre 4. Etude d'un éditeur graphique de restitution

4.1. Introduction

La deuxième étape dans la conception d'un éditeur graphique est la restitution d'un schéma à partir de la base de données des spécifications. Tout comme dans le chapitre précédent, nous allons, tout d'abord étudier des systèmes existants, l'un commercialisé par Arthur Young Proware et l'autre, développé à l'université de Rome par le professeur Batini. La confrontation de ces systèmes nous permettra de définir une architecture fonctionnelle d'un système de restitution.

Ensuite, nous nous sommes concentrés sur la réalisation d'une partie de l'éditeur de restitution : le générateur de brouillon d'un modèle E.R.A. Plusieurs possibilités se sont offertes à nous pour la mise en page d'un schéma. Nous allons les exposer et expliquer les raisons du choix pour lequel nous avons finalement opté.

Nous tenons à signaler que toutes les solutions ont été élaborées avec l'aide de Marcel Clantin, assistant à l'Institut d'informatique.

4.2. Analyse de systèmes existants

4.2.1. Information Engineering Workbench

Comme nous l'avons vu au chapitre précédent, "I.E.W." est un logiciel permettant la saisie et la restitution d'informations par le biais du traitement graphique. Les remarques que nous avons relevées au sujet de l'éditeur de restitution sont partiellement exposées en 3.2.2.3. Ces remarques sont valables aussi bien pour la saisie que pour la restitution des données, à savoir la représentation des noms d'objets et de connexions, la représentation des objets et leur taille.

Pour ce qui est de la restitution proprement dite, notons encore quelques possibilités intéressantes de "I.E.W." et quelques faiblesses :

- Affichage : l'utilisateur peut demander de ne voir apparaître à l'écran qu'une partie de son diagramme de façon à accroître la lisibilité. Les différentes possibilités de simplification sont d'une part l'affichage d'un modèle simplifié, c'est-à-dire qu'on ne représente que certaines informations (par exemple, l'utilisateur ne veut plus voir apparaître les contraintes de quantification dans son modèle Entités-Relations), d'autre part l'affichage d'un sous-modèle, à savoir un objet et tous ses voisins ou deux objets et l'ensemble des chemins pouvant relier ces deux objets et enfin la possibilité d'éliminer les détails (pour une structure arborescence par exemple, suppression de tous les descendants d'un objet).
- Représentation graphique : Le logiciel ne tient aucun compte des croisements : non seulement ils ne sont pas minimisés, mais il peut arriver qu'ils soient cachés par un objet !

Notons également que s'il y a un croisement, les noms des connexions peuvent se chevaucher.

Lors de la restitution d'un diagramme, les objets sont disposés comme ils ont été saisis. S'ils n'ont pas été saisis de façon graphique (ou s'ils ont été saisis via un autre type de diagramme), ils sont disposés le long d'une seule diagonale. Cela signifie, nous a dit le membre de chez Arthur Young Conseil qui a fait la démonstration, que si, "lors de la saisie, vous ne refaites pas régulièrement la mise en page de chaque diagramme, vous en avez pour trois jours avant d'avoir un schéma utilisable !"

Nous pouvons donc en conclure que "I.E.W."n'a pas été étudié comme système de restitution, la saisie texte proprement dite étant impossible. Les avantages que nous avons décelés relèvent plus de l'exploitation d'un diagramme que de sa restitution.

4.2.2. Système de Batini

4.2.2.1. Introduction

Le système de restitution de Batini est actuellement en développement à l'université de Rome [BATI85]. Il fait partie d'un logiciel de grande envergure basé sur le modèle INCOD (INteractive COnceptual Design of Data). INCOD est un environnement permettant la définition des données (via le modèle E.R.A.), la définition des transactions, à différents niveaux d'abstraction et la définition des événements par les réseaux de Pétri.

INCOD gère un ensemble de tâches interactives permettant d'aider l'analyste dans sa conception. On relèvera comme tâches principales :

- Le contrôle automatique de la cohérence

- Un système de dialogue avec l'utilisateur de façon à découvrir les conflits
- L'affichage de différentes solutions possibles de scénarios.

Une extension naturelle à ce logiciel était donc (pour les raisons abordées au chapitre 2) la création d'une interface graphique (figure 4.1.)

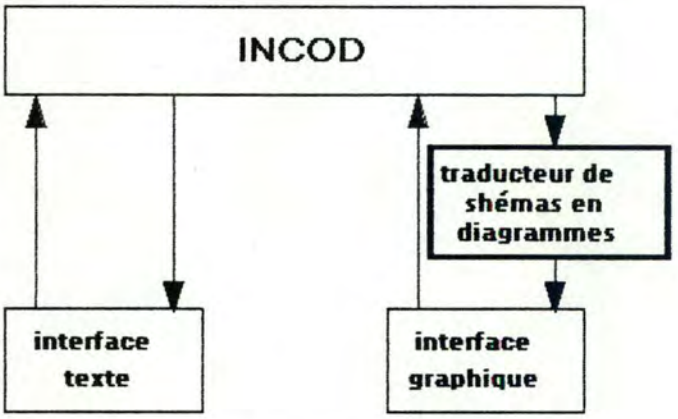


figure 4.1.

La partie qui nous intéresse dans le cadre de ce chapitre est le traducteur de schémas en diagrammes.

4.2.2.2. Description du traducteur

Les différents exemples que nous utiliserons dans le reste de ce paragraphe sont tous basés sur le modèle E.R.A. Batini propose la représentation graphique de la figure 4.2. pour les différents objets du modèle. Notons également que la seule représentation graphique pour un objet est sa forme. Tous les attributs des objets (pour une entité par exemple son nom, son identifiant, ses attributs, sa description, ...) ne sont accessibles qu'en mode texte.

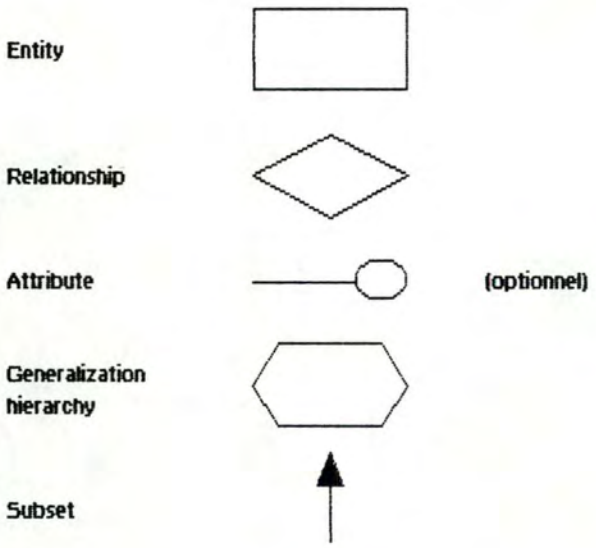


figure 4.2.

L'écran du traducteur se compose de deux fenêtres : d'une part la fenêtre de dessin dans laquelle sera représentée le diagramme et d'autre part la fenêtre des messages dans laquelle se fait le dialogue avec l'utilisateur via un langage de commandes.

Les différentes options choisies pour la représentation d'un diagramme sont les suivantes :

- Le diagramme est représenté dans une grille. Chaque cellule de cette grille peut contenir au plus un objet, mais un objet peut tenir sur plusieurs cases en fonction du nombre de connexions qu'il possède (figure 4.3.)

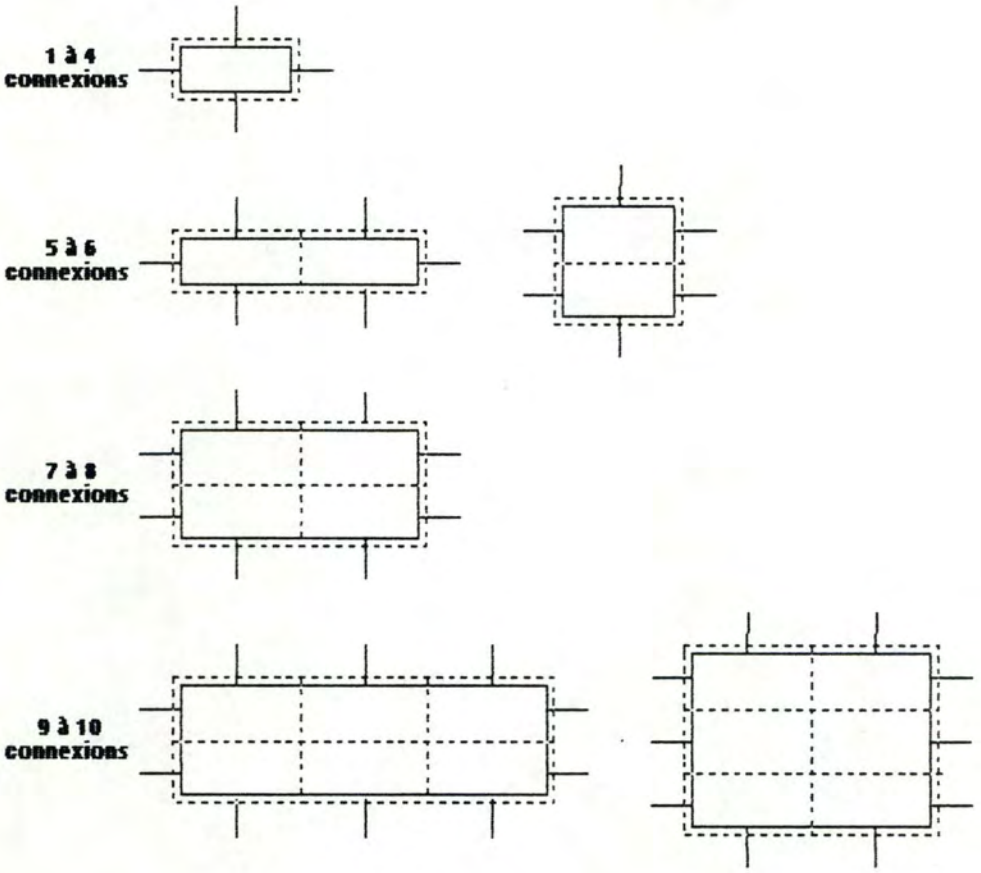


figure 4.3.

- Les connexions entre les objets sont des lignes, horizontales ou verticales situées au centre d'une cellule de la grille. Il ne peut y avoir qu'une seule connexion par case, sauf dans le cas d'un croisement. (figure 4.4.)

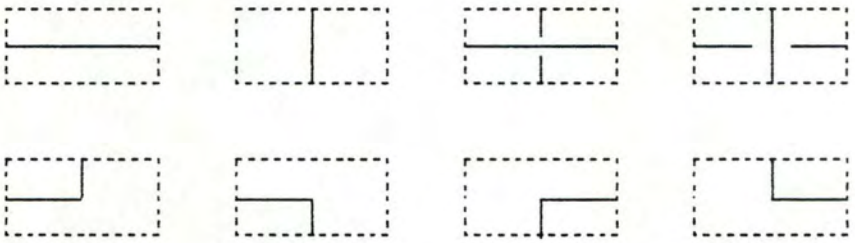


figure 4.4.

- Il existe deux modes de travail : soit la production manuelle d'un diagramme (ou U-mode : User driven mode), soit la production automatique (ou S-mode : System driven mode)

La figure 4.5. présente un exemple de diagramme produit avec le traducteur de schémas.

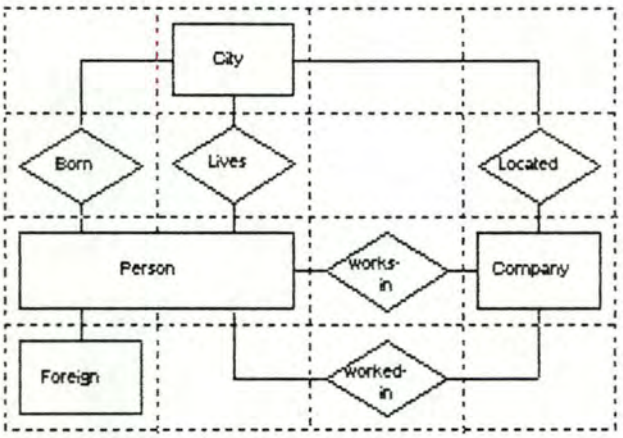


figure 4.5.

4.2.2.3. La production automatique d'un diagramme

a) Concepts à respecter pour avoir un bon diagramme

La lisibilité d'un diagramme dépend de trois aspects :

- La représentation graphique, c'est-à-dire l'aspect du diagramme, indépendamment de sa signification
- La sémantique du modèle, c'est-à-dire la signification des types d'objets utilisés dans le diagramme (par exemple : un objet qui est l'agrégation de plusieurs autres (notion de hiérarchie) doit être placé au-dessus de ceux-ci)
- La sémantique du diagramme, c'est-à-dire la signification d'un diagramme donné (par exemple : l'objet le plus important doit être placé au centre)

Les deux premiers concepts peuvent-être exprimés automatiquement par le système. Le troisième ne peut l'être que via des contraintes introduites par l'utilisateur.

b) Stratégie

Les différentes étapes à respecter dans la construction du traducteur de schémas sont, au vu des concepts cités plus haut :

- La définition d'un standard graphique en fonction de la syntaxe du diagramme,
- L'identification d'un ensemble de règles, permettant d'avoir un diagramme bien ordonné,
- L'ajout de règles permettant de respecter les aspects sémantiques du modèle,
- L'expression de la sémantique du diagramme par des contraintes,
- Un algorithme de dessin qui tient compte des points précédents.

c) Méthodologie

Etant donné l'utilisation d'une grille, les règles concernant l'ordonnancement du diagramme peuvent s'énoncer comme suit :

Règle 1 : Le nombre de croisements doit être minimal

Règle 2 : Le nombre de pliures dans une connexion doit être minimal

Règle 3 : La longueur totale des connexions doit être minimale

Règle 4 : La surface totale du diagramme (taille de la grille) doit être minimale

Remarquons qu'il peut y avoir incompatibilité entre ces quatres règles. La figure 4.6.a représente un diagramme respectant la règle 1, la figure 4.6.b représente ce même diagramme, mais respectant la règle 2.

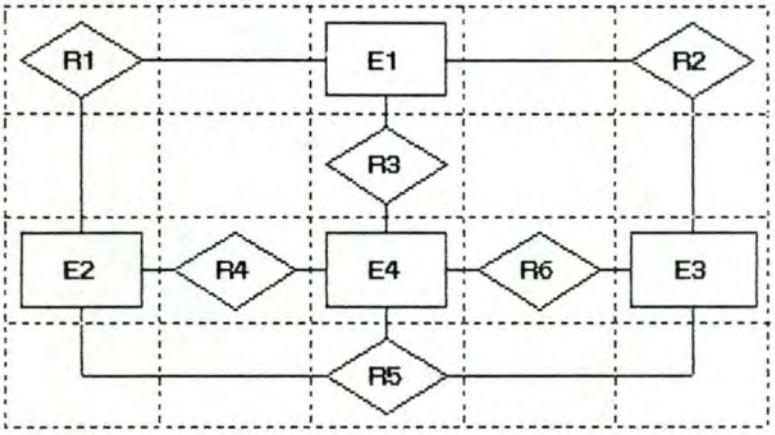


figure 4.6.a.

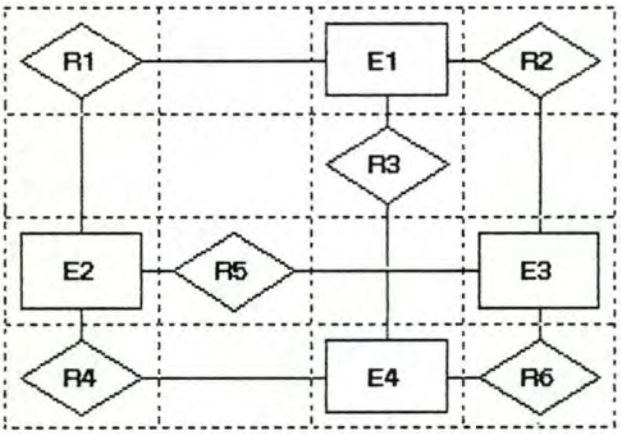


figure 4.6.b.

Etant donné la syntaxe de certains modèles, deux règles peuvent être ajoutées :

Règle 5 : Les structures hiérarchiques doivent être représentées verticalement

Règle 6 : Il doit y avoir symétrie des objets fils dans une hiérarchie, par rapport à leur père.

Les contraintes devant être supportées par le traducteur de schémas sont :

Contrainte 1 : Etant donné un ensemble d'objets spécifiés par l'utilisateur, ceux-ci devront se trouver regroupés (le plus proche possible) sur le diagramme

Contrainte 2 : Un objet donné doit être placé au centre du diagramme.

Les différentes étapes dans la réalisation d'un diagramme peuvent être schématisées comme suit (figure 4.7.) :

A partir du schéma conceptuel, le système détermine toutes les arêtes sur le diagramme ainsi que tous les sommets. Les entités sont toujours des sommets, les relations sont également des sommets si elles sont au moins ternaires.

On obtient donc un graphe conceptuel composé d'un ensemble de sommets V et d'un ensemble d'arêtes E.

Ce graphe est alors rendu planaire puis orthogonalisé, c'est-à-dire que les arêtes sont disposées de manière à n'être plus formées que de lignes horizontales et verticales.

Une fois cette opération terminée, le graphe est placé dans une grille et chaque sommet du graphe est remplacé par la représentation de l'objet qu'il remplace.

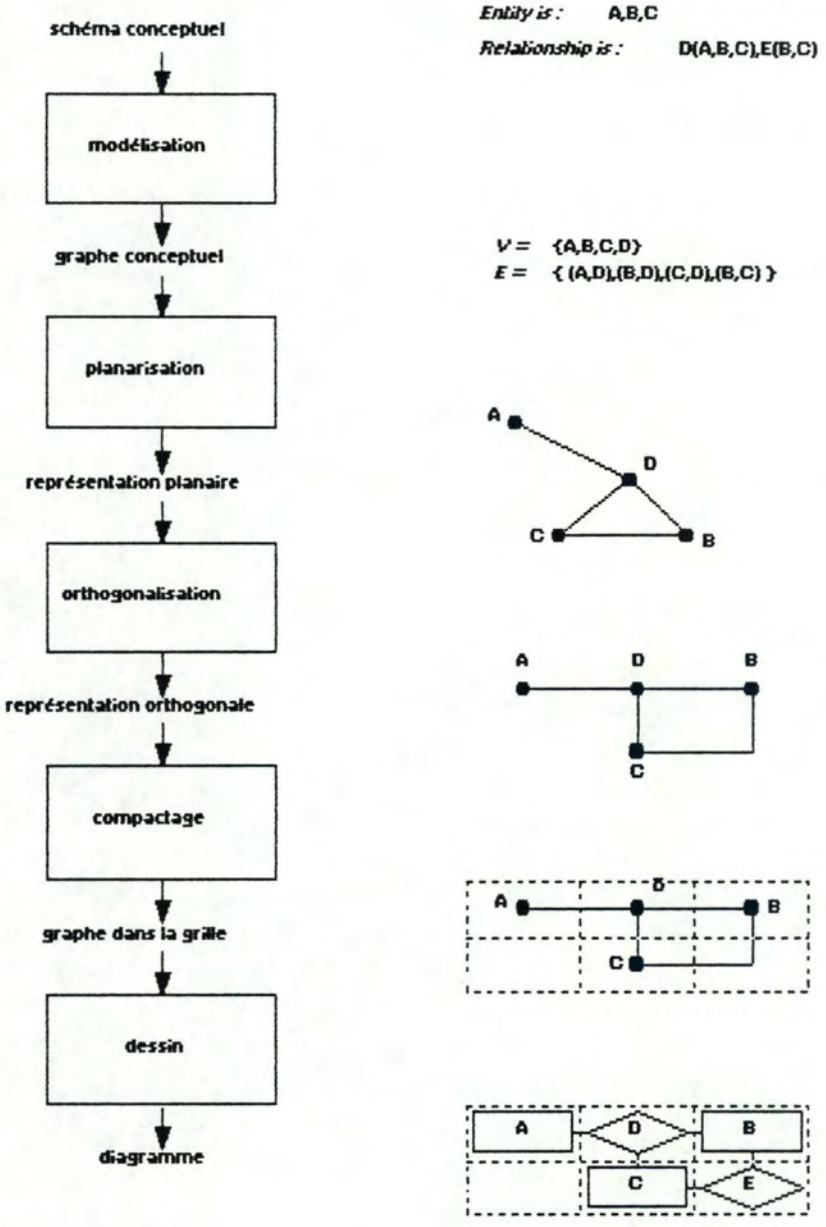


figure 4.7.

Si le lecteur le désire, il pourra trouver des idées d'algorithmes pour l'implémentation du traducteur de schémas en diagrammes dans [BATI83].

4.3. Contribution au développement de l'éditeur graphique dédié à IDA

4.3.1. Architecture fonctionnelle d'un système de restitution

4.3.1.1. Vue générale

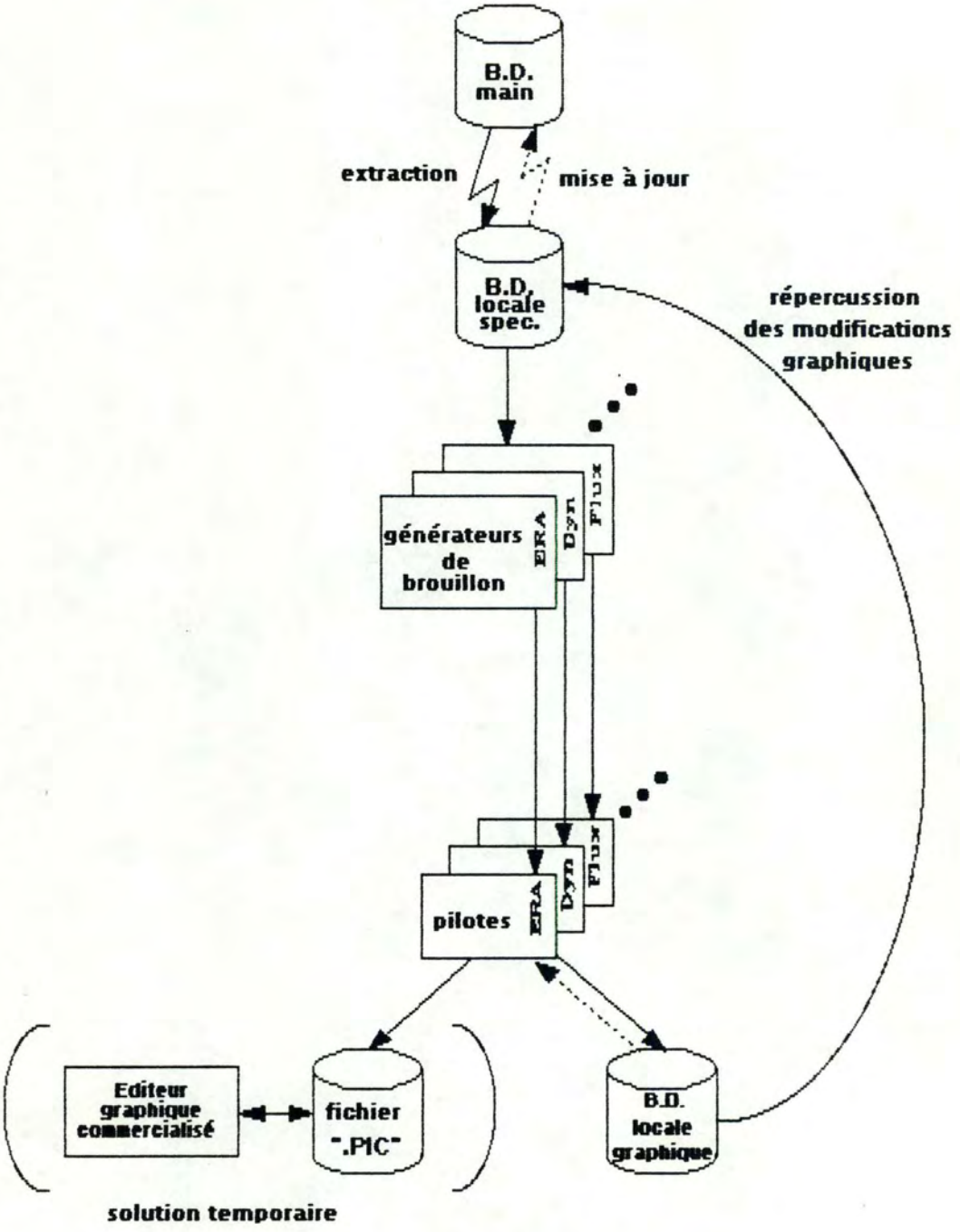


figure 4.8.

Après avoir extrait de la base de données centrale les informations nécessaires et les avoir stockées dans la base de données locale des spécifications, le système invoque le générateur de brouillon associé à l'aspect sur lequel l'utilisateur travaille. Le système va alors proposer un diagramme pour les données contenues dans la base de données locale. Ce diagramme sera ensuite envoyé au pilote qui, avec l'aide de l'utilisateur, va permettre une mise en page définitive.

Dans un premier temps, nous avons pensé créer un fichier ".PIC" permettant de relire ce diagramme via un logiciel graphique commercialisé travaillant par objets dont nous connaîtrions la méthode de stockage des données. Cette solution temporaire permettrait de fournir à l'utilisateur, dans de brefs délais, toute une série de possibilités d'édition et d'impression (zooming, agrandissement du diagramme ou d'un objet du diagramme, dessin d'un objet en trait gras pour le distinguer des autres, suppression d'une partie du schéma, impression de la totalité ou d'une partie du diagramme, ...).

Dans un deuxième temps, le pilote devra être capable d'accepter des modifications de la base de données graphique et de les répercuter dans la base de données locale des spécifications, puis de mettre à jour la base de données centrale. Il devra également être capable d'assurer les fonctionnalités qui étaient, jusque là assurées par l'éditeur graphique commercialisé.

4.3.1.2. L'éditeur de restitution

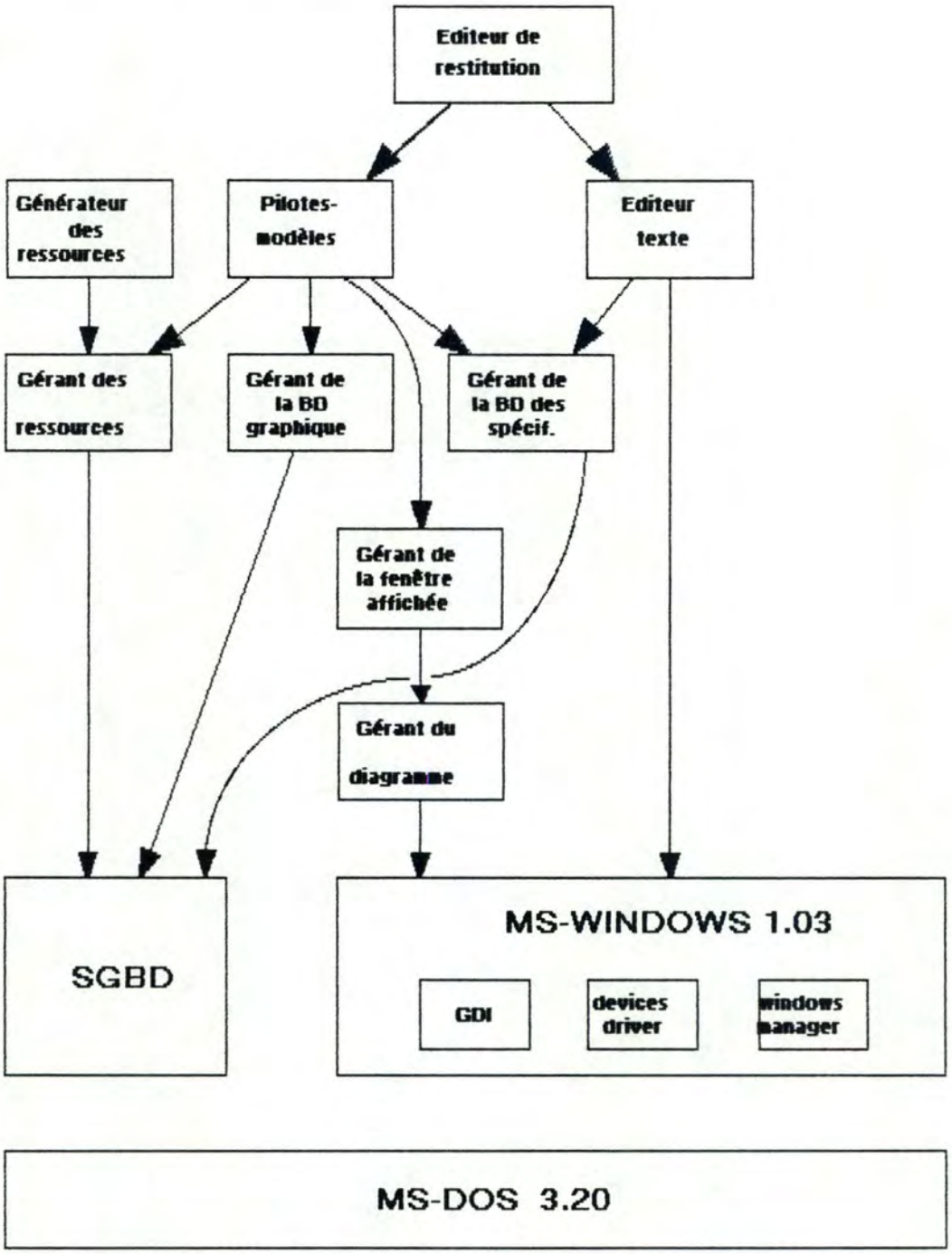


figure 4.9.

L'architecture d'un système de restitution est très semblable à celle d'un système de saisie. Beaucoup de modules peuvent même être communs aux deux architectures (tous les modules des niveaux 0 à 4, le générateur des ressources et l'éditeur de texte). Cette architecture est présentée à la figure 4.9.

niveau 0 : O.S. : Le système d'exploitation MS-DOS 3.20

niveau 1 : Le système de gestion de base de données.

MS-WINDOWS 1.02 ou 1.03 : logiciel de gestion de fenêtres (bureau électronique), de menus, ..., logiciel de gestion graphique (GDI : Graphics Display Interface), gérant des périphériques (imprimantes, tables traçantes, cartes graphiques, souris, ...)

niveau 2 : Le gérant du diagramme : module ayant pour fonction la gestion du diagramme dans le monde graphique. Le monde graphique est un espace de travail indépendant du mode de résolution et des spécificités de l'espace physique.

niveau 3 : Le gérant de la fenêtre affichée : ce module a pour fonction le zooming et la gestion de la fenêtre dans le monde graphique.

niveau 4 : Le gérant des ressources : ce module doit assurer la gestion de la représentation graphique, à savoir, la représentation des objets graphiques configurée par l'utilisateur (il choisit ce qu'il veut représenter et comment il veut le représenter).

Le gérant des entrées / sorties sur bases de données : il est composé de deux modules, à savoir la mémorisation des graphiques et la mémorisation des spécifications au niveau local

niveau 5 : Le générateur des ressources : c'est une interface qui permet à l'utilisateur de spécifier, de façon simple et agréable la liste des objets qu'il veut voir apparaître en mode graphique et leur représentation à l'écran.

L'éditeur de texte : c'est le composant permettant la restitution des informations n'ayant pas de représentation graphique.

Les pilotes sont les modules destinés à aider l'utilisateur à mettre en page de la façon la plus esthétique à ses yeux les informations graphiques contenues dans la base de données. La restitution des données peut se faire de façon manuelle (l'utilisateur place lui même les objets dans la page graphique), de façon semi-automatique (le système place lui même les objets avec une forte interaction de l'utilisateur) ou de façon automatique.

niveau 6 : L'éditeur de restitution est composé des pilotes et de l'éditeur texte.

4.3.2. Evolution

Lors de l'élaboration du générateur de brouillon pour le modèle E.R.A., nous avons eu de nombreuses discussions, et des solutions ou ébauches de solutions en ont découlé. Nous retraçons ici les étapes les plus importantes ainsi que les raisons pour lesquelles nous n'avons pas poursuivi dans ces voies, pour finalement aboutir à la solution que nous avons retenue.

4.3.2.1. Génération manuelle, création semi-automatique d'un environnement simple

Nous définissons l'environnement simple d'un objet comme l'ensemble des objets qui sont directement liés à cet objet par une connexion (par exemple, l'ensemble des associations reliées à une entité).

Nous avons constaté qu'un modèle E.R.A. possède un objet central, c'est à dire un objet qui a plus de connexions que les autres. Il semble donc que c'est pour cet objet que nous aurons le plus de difficultés, car il risque d'être relié à d'autres, fort éloignés. La solution que nous avons adoptée est de le placer dans le diagramme le premier. A partir de cet objet, le placement de son environnement simple peut se faire de manière automatique, mais l'utilisateur a la possibilité à tout moment de déplacer n'importe quel objet pour arranger le diagramme à sa guise.

Pour plus de facilité, nous présentons cette solution sur base d'un exemple.

A partir de l'extraction de la base de données centrale (représenté par le schéma de la figure 4.10.), nous construisons les tables représentant les relations "relates" et "related by" (figure 4.11.).

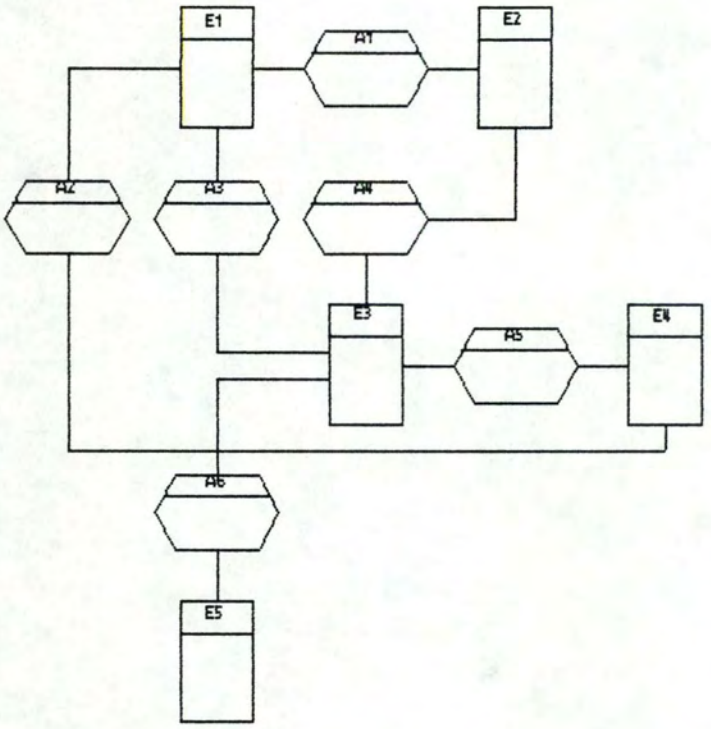


Figure 4.10.

Related by		Relates	
E5 :	A6	A1 :	E1
			E2
E1 :	A1		
	A3	A2 :	E1
	A2		E4
E2 :	A1	A3 :	E1
	A4		E3
E3 :	A5	A4 :	E3
	A3		E2
	A4		
	A6	A5 :	E3
			E4
E4 :	A5		
	A2	A6 :	E3
			E5

Figure 4.11.

Nous sélectionnons l'objet ayant le plus de connexions (E3 dans l'exemple). Nous plaçons l'objet au centre de l'écran ainsi que son environnement simple (A3, A4, A5 et A6 dans l'exemple).

Remarque : dans le cas de 4 connexions pour un objet placé, chaque objet de l'environnement simple se place aux quatre points cardinaux; dans le cas de deux connexions, les objets se placent à l'opposé l'un de l'autre, mais dans le cas de trois connexions, un choix se pose. Nous avons opté pour mettre l'objet de l'environnement simple ayant le plus de connexions entre les deux autres (par exemple en bas si on a placé les deux autres respectivement à gauche et à droite). En effet, cet objet est le plus susceptible d'être relié à d'autres objets déjà placés sur l'écran.

Pour chaque objet de l'environnement simple de l'objet analysé, nous positionnons son propre environnement simple en tenant compte des objets déjà positionnés sur l'écran. Dans l'exemple, lorsque nous plaçons A1 comme environnement simple de E2, nous le mettons à sa gauche car il est relié à E1 (voir figure 4.12.).

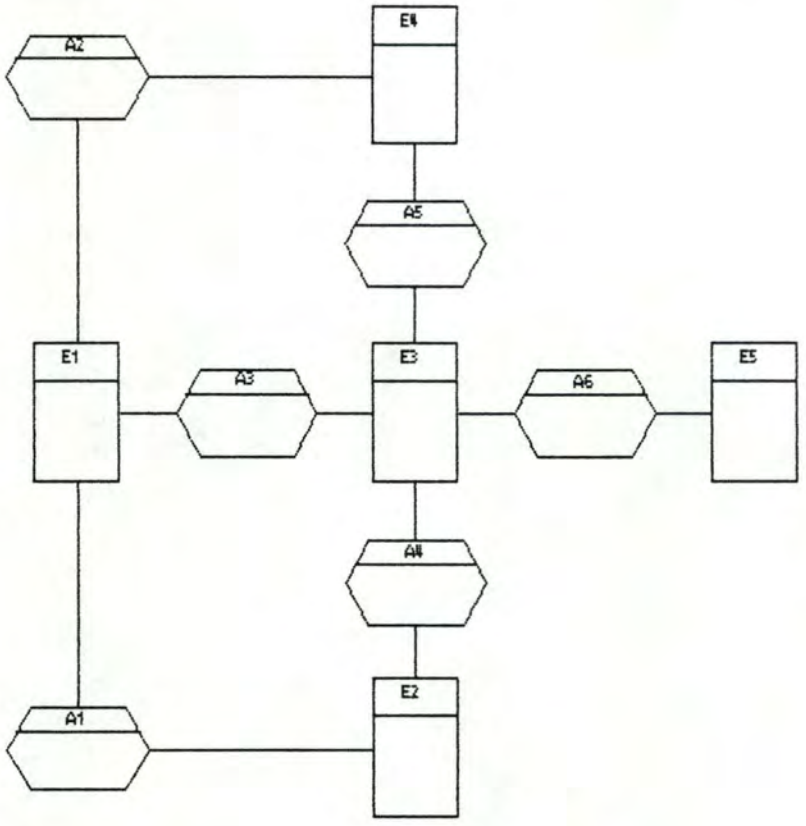


Figure 4.12.

Nous itérons le processus de positionnement de l'environnement simple d'un objet en parcourant les objets "en largeur d'abord", c'est à dire en positionnant l'environnement simple de tous les objets appartenant à l'environnement de l'objet central d'abord. (nous faisons en fait de la récursivité).

Condition de terminaison : le modèle E.R.A. est totalement représenté quand nous avons dessiné autant de connexions qu'il n'y a de "relates" (une des conditions possibles).

Après le placement de chaque objet, l'utilisateur a le loisir de les déplacer à son gré. Pour ce faire, nous avons besoin des primitives suivantes :

inversion de deux éléments (éventuellement de deux groupes d'éléments)

déplacement d'un groupe d'objets.

Inconvénients de la solution :

Après avoir testé la méthode avec succès sur des graphes simples (une dizaine d'objets) nous avons essayé un graphe plus complexe (plus proche d'un modèle réel : figure 4.13.)

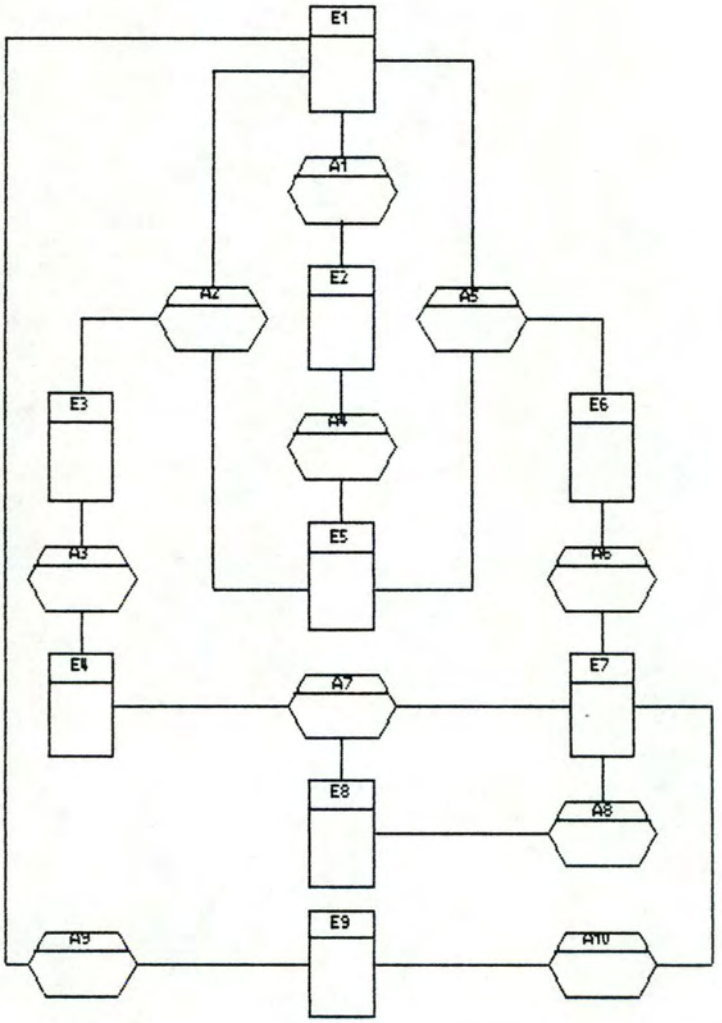


Figure 4.13.

La construction des tables (figure 4.14.) nous suggère de commencer par l'entité E1 ou l'entité E7.

Related by		Relates	
E1 :	A1	A1 :	E1
	A2		E2
	A5		
	A9	A2 :	E1
			E5
E2 :	A1		E3
	A4		
		A3 :	E3
E3 :	A2		E4
	A3		
		A4 :	E2
E4 :	A3		E5
	A7		
		A5 :	E1
E5 :	A2		E5
	A4		E6
	A5		
		A6 :	E6
E6 :	A5		E7
	A6		
		A7 :	E4
E7 :	A6		E7
	A7		E8
	A8		
	A10	A8 :	E7
			E8
E8 :	A7		
	A8	A9 :	E9
			E1
E9 :	A10		
	A9	A10 :	E9
			E7

Figure 4.14.

Le problème s'est posé lors du placement du 17ème élément (sur 19). En effet, il n'y a pas moyen, en utilisant des primitives simples, de placer A7 sans provoquer de croisement (figure 4.15.). Or, il nous semble qu'une

des qualités d'un éditeur de restitution dédié au modèle E.R.A est de minimiser les croisements.

Un autre problème de taille est le manque d'une vue globale du diagramme pour l'utilisateur : lorsqu'il place un objet, l'utilisateur n'a aucune idée de l'aspect final de son schéma. Il nous a semblé qu'il s'agissait là d'une lacune fondamentale.

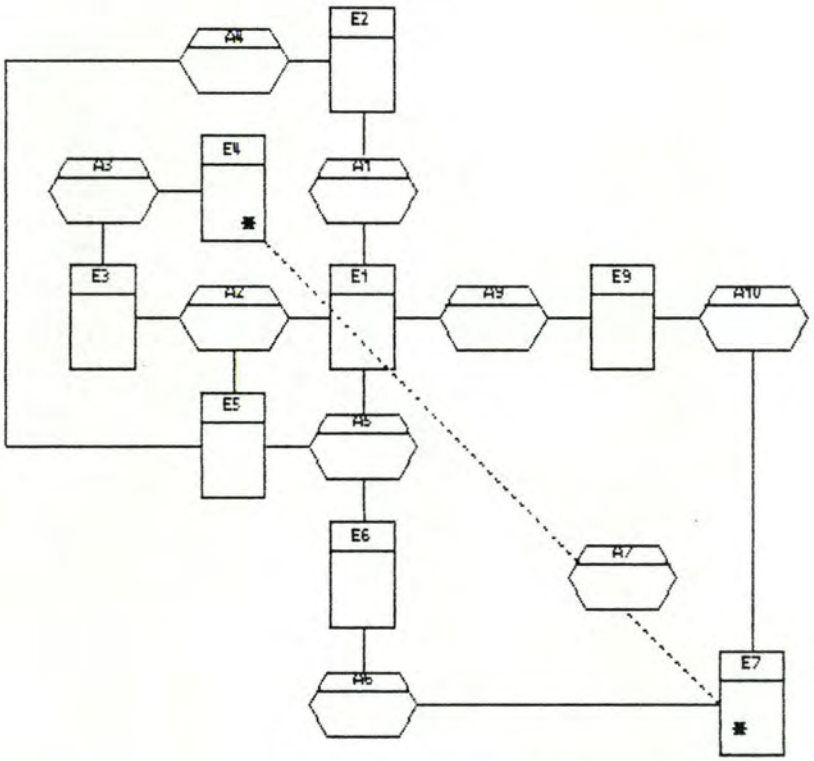


Figure 4.15.

Ces problèmes nous ont donc incités à trouver une autre méthode. Elle se rapproche plus du générateur de brouillon, car elle propose d'emblée une vue du diagramme complet.

4.3.2.2. Génération semi-automatique

Le principe de cette solution est de sélectionner l'objet possédant le plus de connexions, et à partir de celui-ci, de construire le modèle E.R.A. sous forme d'un graphe découpé en niveaux. Le diagramme ainsi obtenu est présenté à l'utilisateur. Si le graphe contient des croisements, l'utilisateur peut supprimer une partie du schéma. Les objets ainsi supprimés seront réintroduits manuellement.

Pour chaque entité, nous reprenons toutes les associations qui lui sont connectées et pour chaque association toutes les entités qui lui sont connectées (figure 4.14.).

Nous choisissons dans ces deux listes l'entité ou l'association qui a le plus grand nombre de connexions (E1 ou E7 dans l'exemple).

A partir de l'objet choisi, nous construisons un graphe par niveau (figures 4.16. et 4.17.).

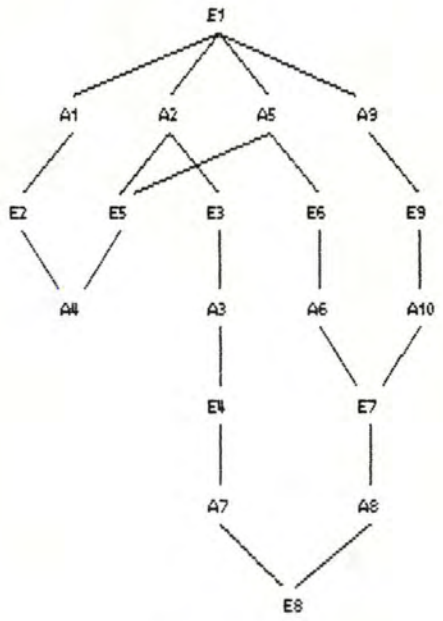


Figure 4.16.

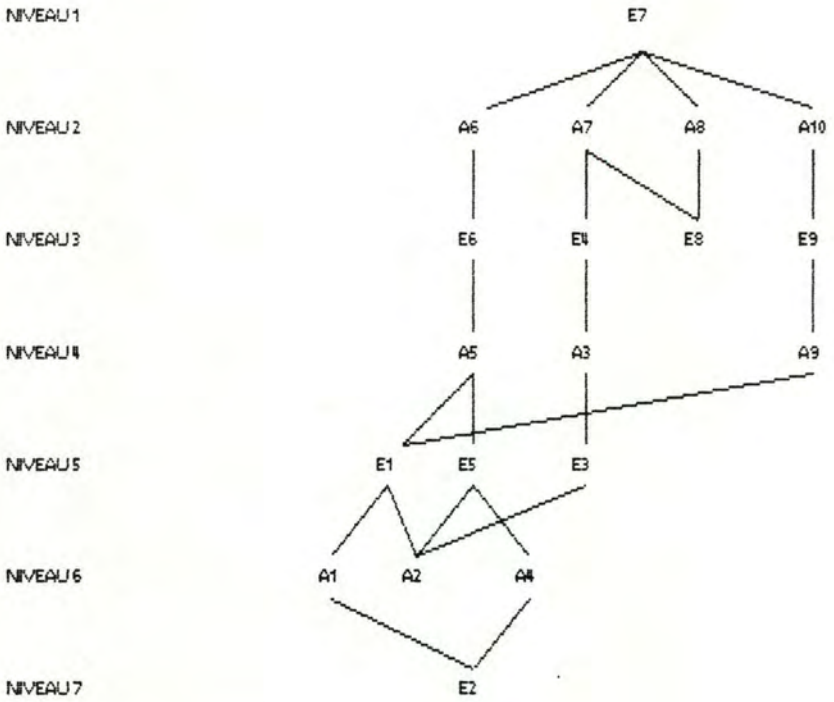


Figure 4.17.

Si l'objet choisi est une entité, le deuxième niveau comprendra toutes les associations qui y sont connectées, le troisième niveau les entités qui sont connectées aux associations du niveau précédent si elles n'apparaissent pas au premier niveau, etc ...

Si l'objet choisi est une association, le deuxième niveau comprendra toutes les entités qui y sont connectées, et ainsi de suite.

Une fois le graphe terminé, nous en déduisons le modèle E.R.A. (le dessin correspond exactement au graphe) (Les figures 4.17. et 4.18. sont en fait le même graphe).

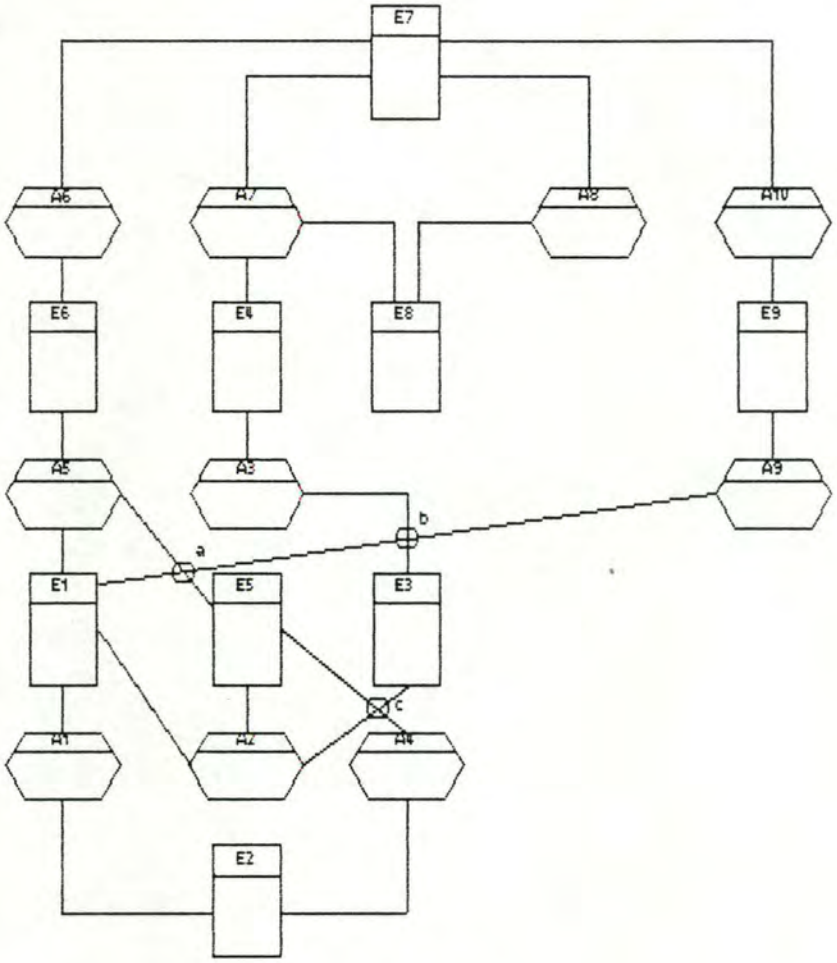


Figure 4.18.

Si le modèle E.R.A. ainsi obtenu ne contient pas de croisement, nous pouvons considérer que le résultat de l'algorithme est satisfaisant. L'utilisateur pourra réarranger la position des objets à sa guise.

Si le modèle E.R.A. contient des croisements, l'utilisateur pourra appliquer les primitives de base (move, ...). Il aura également la possibilité de supprimer des objets de son diagramme (au minimum deux). L'utilisateur aura alors à l'écran un modèle E.R.A où les objets supprimés et tout ce qui s'y rapporte (connexions et objets "solitaires" dus à la suppression de certaines connexions) n'apparaîtront plus. Il pourra ainsi se rendre compte s'il ne subsiste aucun croisement (si c'est le cas, il pourra

sélectionner des objets supplémentaires ou "défaire" ses sélections pour en faire des nouvelles). Dans l'exemple (figure 4.11.), l'utilisateur pourrait déplacer le groupe d'objets (A10, E9, A9) vers la gauche (suppression des croisements 'a' et 'b') et supprimer les objets (E5, E3, A2, A4) ce qui provoque la suppression du croisement 'c'. Il ne s'agit ici que d'un scénario possible. L'utilisateur est responsable du choix des objets à supprimer. Il n'est pas aidé par le système.

Remarque : il aurait pu se contenter de supprimer E5 et E3 par exemple.

Une fois ces opérations effectuées, le système reconstruira le graphe en partant du niveau le plus éloigné des niveaux auxquels les objets ont été supprimés, en oubliant les objets sélectionnés et proposera à l'utilisateur un nouveau modèle incomplet.

A partir de ce modèle incomplet, le système proposera à l'utilisateur les objets supprimés un à un. L'utilisateur les placera lui-même sur le dessin. Le programme lui indiquera les autres objets auxquels cet objet doit être connecté (par exemple en les dessinant en blanc sur fond noir). L'utilisateur "n'aura plus qu'à" le placer de manière à ne plus avoir de croisements (figure 4.19.). Encore une fois, il est libre de placer l'objet où il le désire, il n'est pas aidé par le système.

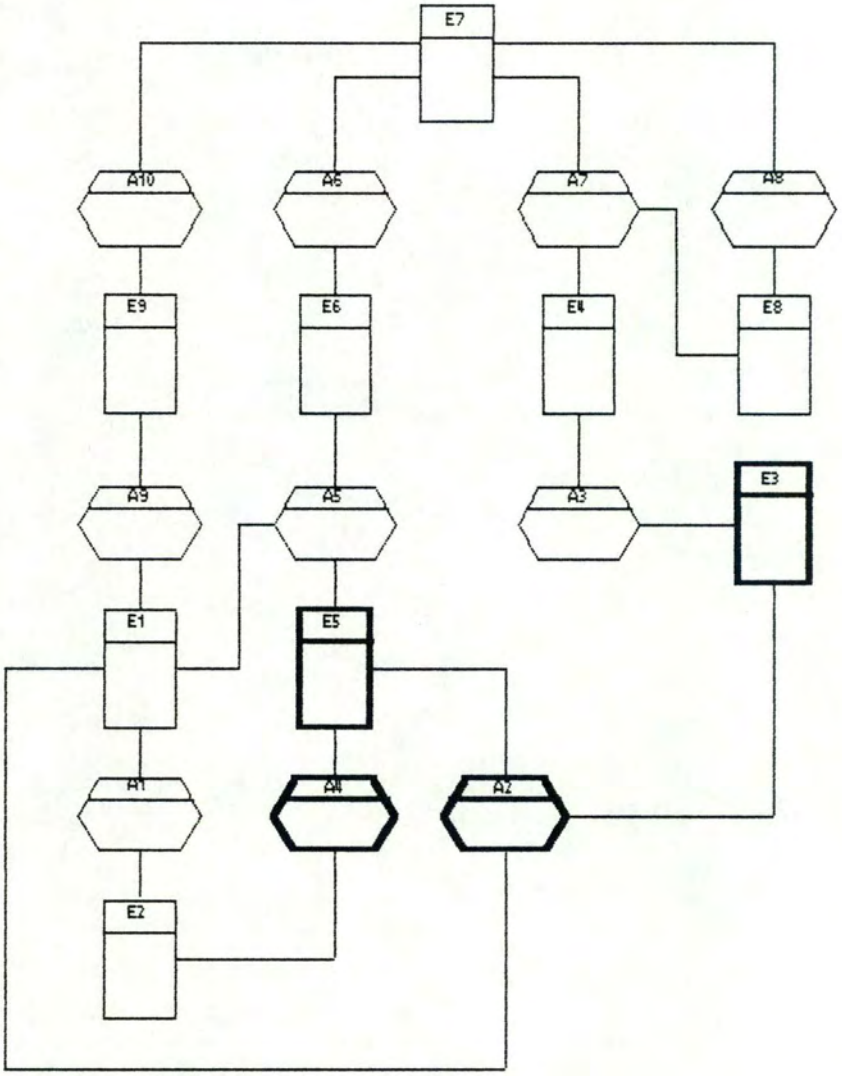


Figure 4.19.

Les figures 4.20. et 4.21. montrent l'application de la méthode à un exemple plus complexe.

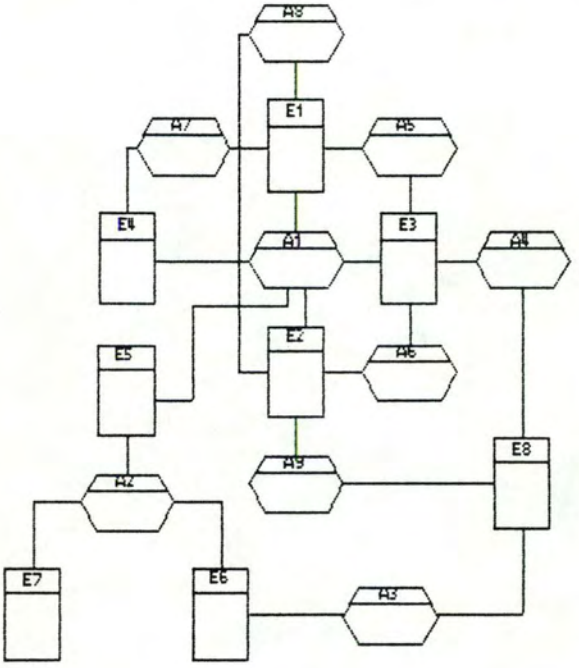


Figure 4.20.

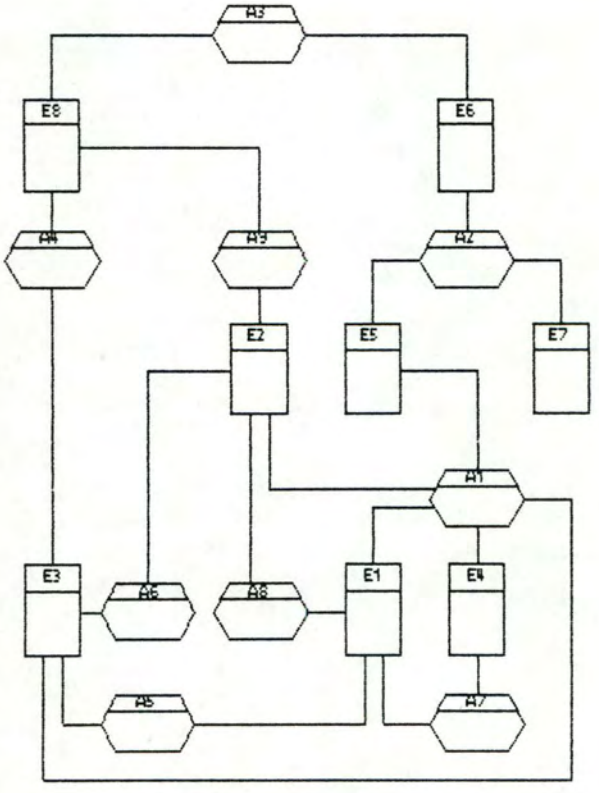


Figure 4.21.

4.3.2.3. Amélioration : détection de la planarité

Pour améliorer cet algorithme, nous avons ensuite tenté de trouver une méthode permettant de signaler à l'utilisateur si le graphe était représentable sans aucun croisement, c'est-à-dire trouver un algorithme capable de détecter la planarité d'un graphe.

Un graphe planaire topologique est un graphe situé dans R^2 dont les sommets sont des points distincts et tel que deux arêtes ne se rencontrent jamais en dehors de leurs extrémités.

On dira qu'un graphe est planaire s'il est possible, en disposant ses sommets judicieusement, de le rendre planaire topologique [ROY69].

La détection de la planarité d'un graphe est un travail très coûteux en temps. Pour diminuer celui-ci, nous avons découvert qu'il était possible de simplifier le graphe en supprimant certains de ses sommets et arêtes sans pour cela modifier sa planarité. Le graphe simplifié est plus facile à analyser pour savoir s'il est planaire. Les sommets et arêtes supprimés peuvent être réintroduits par la suite sans provoquer de croisements puisque les suppressions effectuées n'affectent en rien la planarité du graphe.

Les différentes étapes de la simplification, inspirées de [DEO74] sont les suivantes :

Etape 1 : Une composante connexe d'un graphe est un sous-graphe tel que, pour tout X, Y appartenant à l'ensemble des sommets du sous-graphe, il existe un chemin permettant d'atteindre Y en partant de X et vice-versa. Un graphe est dit connexe s'il existe toujours un chemin permettant de relier deux quelconques de ses points.

Un graphe est planaire si chacune de ses composantes connexes est planaire.

Il reste donc à analyser chaque composante, une fois le graphe décomposé en ses composantes connexes.

Etape 2 : L'ajout ou la suppression d'un sommet pendant n'affectant pas la planarité, nous supprimons ces sommets (figure 4.22).

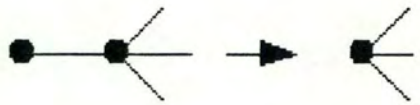


figure 4.22.

Etape 3 : L'ajout ou la suppression d'une arête récursive n'affectant pas la planarité, nous supprimons ces arêtes (figure 4.23.).

figure 4.23.

Etape 4 : Etant donné que les arêtes parallèles ne modifie pas la planarité, nous éliminons les arêtes dédoublées en supprimant toutes les arêtes entre deux sommets, sauf une (figure 4.24.).

figure 4.24.

Etape 5 : La suppression de sommets binaires ne change rien à la planarité. Nous les éliminons donc aussi (figure 4.25.).

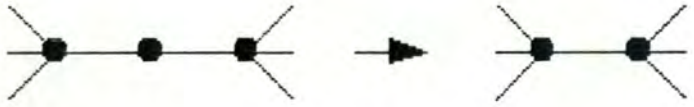


figure 4.25.

Si nous réitérons les étapes 3, 4 et 5, le graphe se réduira souvent considérablement. Un graphe ainsi simplifié se composera soit d'une simple arête (condition minimale de terminaison), soit d'un graphe complet de quatre sommets (dont tous les sommets sont reliés entre eux), ou encore d'un graphe simple, non-séparable, d'au moins cinq sommets et d'au moins sept arêtes.

Il faudra évidemment mémoriser les sommets et arêtes supprimés ainsi que l'ordre de suppression pour pouvoir les réinsérer dans le graphe simplifié après l'analyse.

Les seuls algorithmes que nous avons pu trouver dans la littérature indiquaient dans certains cas la non-planarité. Mais, le fait de ne pas pouvoir démontrer qu'un graphe est non-planaire n'implique pas que celui-ci soit planaire [JOHN72]. Dans certains cas très précis, il est possible de détecter la planarité d'un graphe, mais il n'est malheureusement pas possible de les généraliser en un temps de calcul raisonnable. Nous pensions pouvoir exploiter notre deuxième solution en indiquant à l'utilisateur si son graphe était planaire ou non. Il aurait donc su s'il était possible, via diverses étapes, de supprimer tous les croisements. Il n'aurait donc pas cherché en vain. Mais puisqu'il était impossible de lui

fournir à coup sûr cette information, nous avons abandonné notre recherche d'algorithmes.

4.3.2.4. Solution adoptée

Pour notre quatrième solution, nous avons gardé l'idée de la simplification, de manière à présenter le graphe simplifié à l'utilisateur comme première ébauche. L'utilisateur peut en modifier la mise en page à sa guise. Pour la présentation du graphe simplifié, nous avons repris le principe de base de l'éditeur de restitution de "I.E.W.", à savoir disposer les éléments le long d'une diagonale, en lui apportant certaines améliorations. Avant de les expliquer, nous devons définir certains concepts, propres à la théorie des graphes : [JOHN72] et [DEO74]

Un pivot est un sommet d'un graphe connexe qui peut être divisé en deux ou plusieurs sommets de façon à obtenir un graphe qui n'est plus connexe.

Un pont est une arête d'un graphe connexe telle que, si on la supprime, le graphe n'est plus connexe. Un pont n'est rien d'autre qu'une arête entre deux pivots (exemple figure 4.26.).

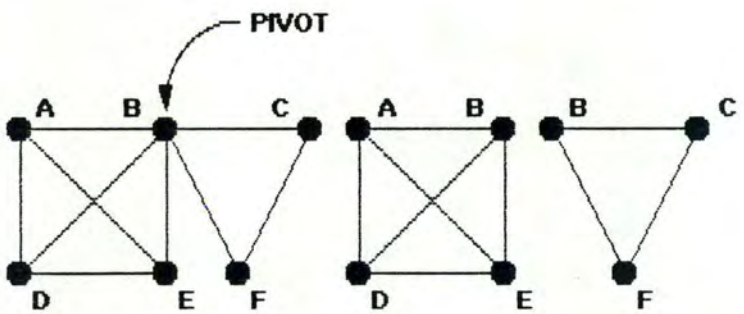


figure 4.26.

Dans un graphe, nous avons remarqué que les objets clefs sont les pivots. En effet, le graphe s'étend autour de ceux-ci. Nous avons donc pris l'option d'utiliser cette propriété pour améliorer la méthode de la diagonale (développée par Arthur Young Proware pour le logiciel I.E.W.), et plutôt que de n'utiliser qu'une seule diagonale, nous en utiliserons autant que la suppression des pivots ne crée de sous-graphes connexes. La figure 4.27. montre un exemple d'articulation d'un graphe autour de ses pivots.

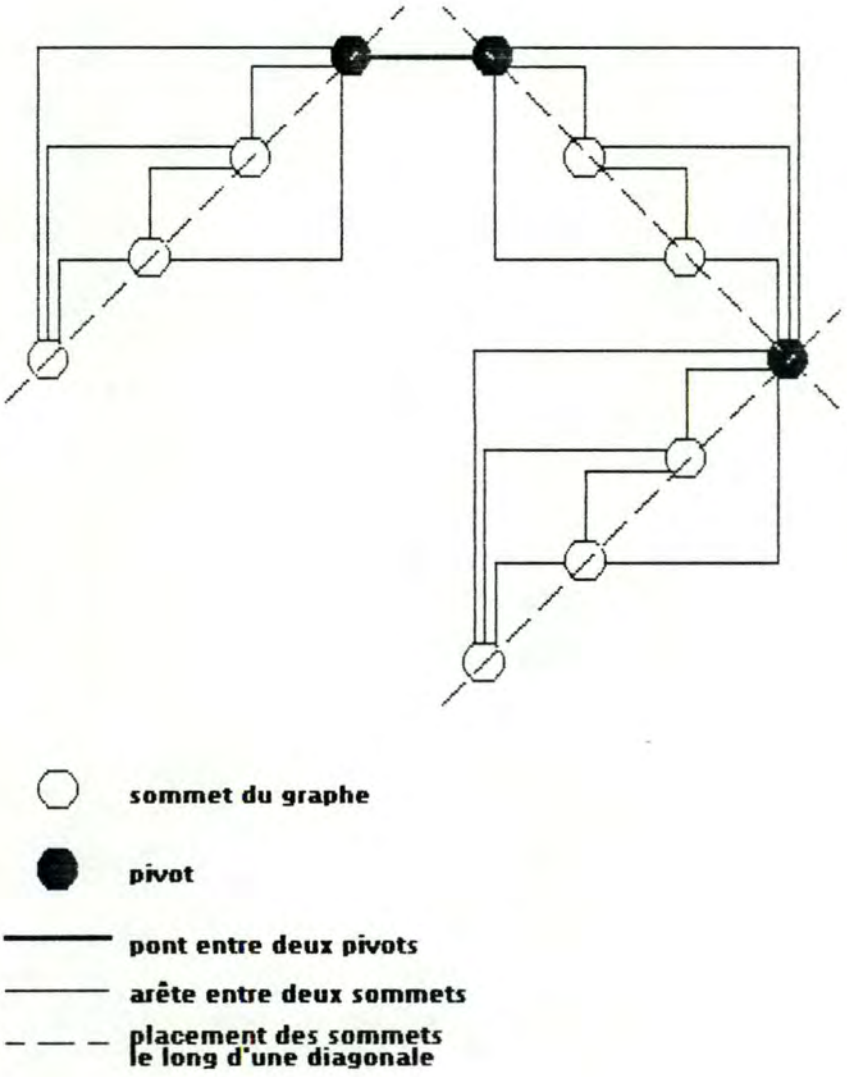
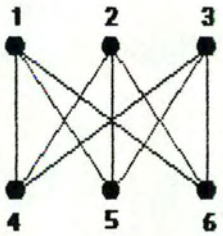


figure 4.27.

Si nous n'avons pas pu démontrer l'efficacité de cet algorithme, de tous les cas testés, du plus simple au plus compliqué, un seul ne s'est pas montré concluant.

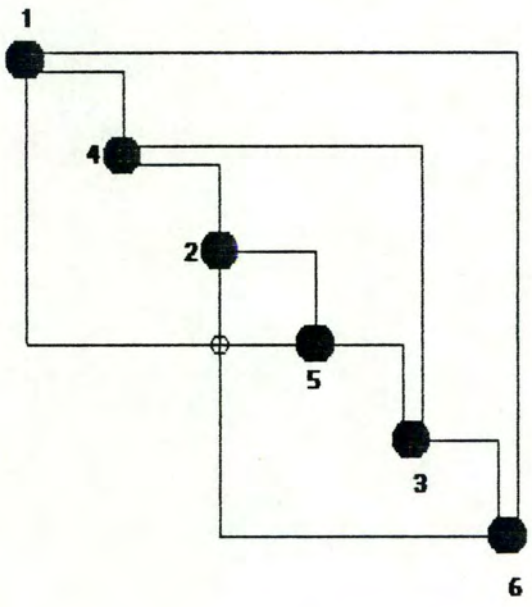
Les figures 4.28. et 4.29. présentent les deux graphes non planaires de Kuratowski.

Les Centrales : (non planaire)



8 croisements

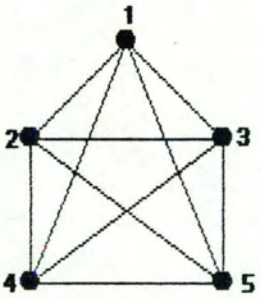
figure 4.28.a



1 croisement

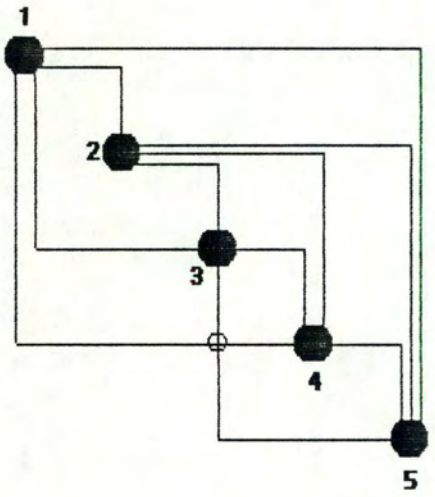
figure 4.28.b

Le pentagone : (non planaire)



5 croisements

figure 4.29.a



1 croisement

figure 4.29.b

La figure 4.30. présente le cas du graphe planaire que nous avons testé et qui a été représenté avec un croisement. La ligne en pointillé représente le chemin qu'aurait du suivre l'arête qui provoque le croisement de manière à rendre le graphe topologiquement planaire.

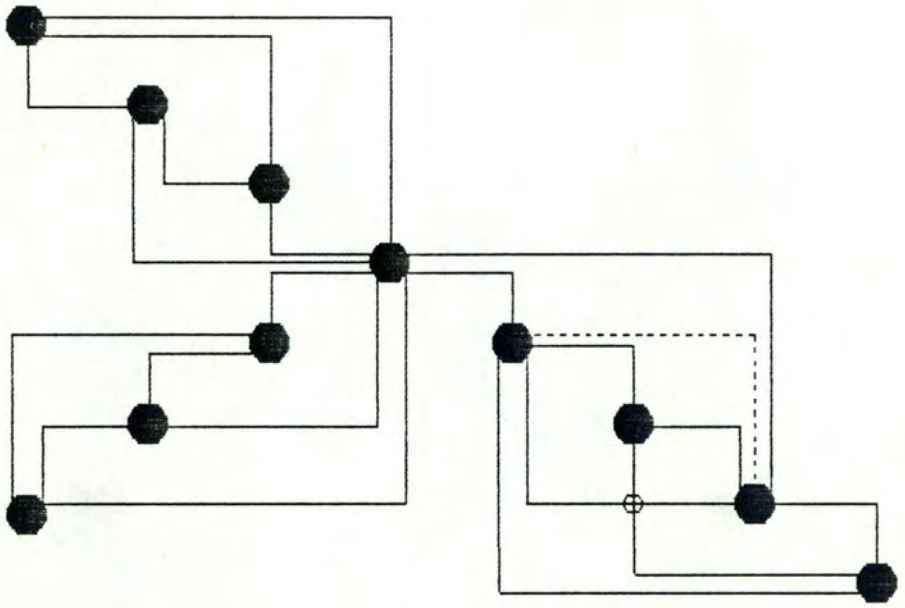


figure 4.30.

La détection des pivots est assez simple et peu coûteuse. Le lecteur trouvera un excellent algorithme dans [DEO74]. Il est exposé dans les spécifications de la fonction de détection des ponts et des pivots.

Quand l'utilisateur a fini d'éditer le graphe simplifié, il reste à réinsérer les sommets et arêtes supprimés. Cette étape peut se faire automatiquement, sans interaction car il est certain que cette réinsertion ne provoque aucun croisement supplémentaire (sauf éventuellement la réinsertion d'une arête dédoublée si un croisement existait déjà au niveau du graphe simplifié). L'utilisateur peut ensuite éditer à nouveau le graphe complet pour l'agencer à sa meilleure convenance.

4.4. Conclusion

Si différents systèmes de saisie graphique existent sur le marché, il n'existe pas de vrais éditeurs de restitution, la plupart d'entre eux étant basés sur des informations graphiques existantes (mémorisées à la saisie des informations). Seul Batini propose réellement un système de restitution complet, c'est-à-dire la création de diagrammes avec, comme seule information de départ, le contenu de la base de données des spécifications. Le système de Batini semble, d'après sa littérature, être efficace, mais les seuls exemples présentés sont des cas rudimentaires (quelques objets) alors qu'un modèle réel est composé d'un nombre important d'objets. Le processus de planarisation est un problème très coûteux en temps calcul. Le processus de planarisation appartient à la troisième classe de la théorie de la complexité, celle qui regroupe les problèmes les plus difficiles. Il semble donc nécessaire de trouver un algorithme qui serait beaucoup plus performant tout en conservant un nombre de croisements, sinon minimal, au moins très réduit.

La solution que nous avons choisie, à savoir la disposition d'un graphe simplifié en diagonale, la possibilité de l'éditer et puis la restitution du graphe complet avec de nouveau possibilité de l'éditer, nous parait la plus adéquate pour un éditeur graphique.

En effet, la première solution (manuelle, celle-ci) soutenait l'utilisateur dans sa tâche mais ne permettait pas une vue globale du diagramme.

La deuxième (affichage du graphe complet avec possibilité pour l'utilisateur de supprimer certains objets pour essayer d'éliminer les croisements) nous a amené à considérer le problème de la minimisation des croisements dans un graphe et sa planarité.

La troisième solution (recherche de la planarité d'un graphe) était très coûteuse en ce qui concerne le temps de calcul. Elle était donc à rejeter d'office vu que nous

n'avions, de plus, trouvé aucun algorithme ou ébauche d'algorithme qui déterminait de façon exacte la planarité d'un graphe..

La solution finale soumet le diagramme à l'analyste à deux reprises, d'abord sous la forme du graphe simplifié, à un niveau d'abstraction très bas et puis sous la forme du schéma reconstitué. Il a en donc une vue plus globale et peut mieux se rendre compte de l'ampleur du dessin qu'il est en train de concevoir.

Mais en plus de cette solution, il faudrait pouvoir proposer à l'utilisateur une option qui lui permettrait de réaliser la mise en page de son diagramme sans l'aide d'aucun pilote. En effet, il pourrait avoir une idée bien précise de son diagramme et pourrait être perturbé de le découvrir sous une autre forme.

Chapitre 5. Conception et implémentation d'un pilote pour la production d'un modèle E.R.A.

5.1. Introduction

Nous arrivons à présent au coeur technique du générateur de brouillon. Nous vous présenterons dans ce dernier chapitre la découpe en sous-modules du générateur. Ces sous-modules seront spécifiés et nous exposerons le format des données ainsi que leurs transformations au cours de l'exécution.

Nous terminerons ce chapitre en rappelant les étapes qui ont été développées et celles encore à implémenter.

5.2. Décomposition du module "générateur de brouillon"

Le générateur de brouillon peut être décomposé en trois parties qui doivent être exécutées consécutivement (figure 5.1.) :

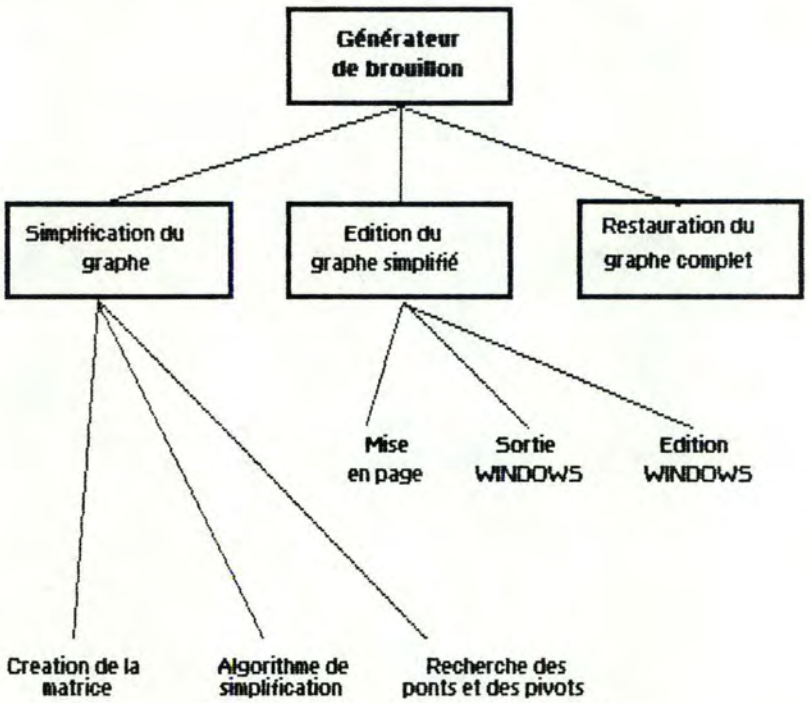


figure 5.1.

La première de ces trois parties consiste à simplifier le graphe après l'avoir saisi dans la base de données locale et à détecter les ponts et les pivots existants dans ce graphe simplifié.

La seconde partie conduit à la première interaction avec l'utilisateur : le graphe simplifié est mis en page, affiché dans une fenêtre "MS-WINDOWS" et peut être simplifié.

Enfin, la troisième partie reconstruit le graphe initial en tenant compte de la disposition des sommets du graphe simplifié.

5.3. Spécifications

5.3.1. Introduction

Les spécifications externes que nous allons présenter sont celles des modules et sous-modules présentés au paragraphe précédent. Les spécifications internes sont celles des fonctions principales de chacun de ces modules. Le lecteur intéressé trouvera les spécifications de chaque fonction dans les programmes, en annexe.

5.3.2. Spécifications externes

MODULE : Générateur de brouillon

INPUT : Graphe complet dans la base de données locale des spécifications

OUTPUT : Base de données locale graphique

TRAITEMENT : Constitution de la base de données locale graphique en simplifiant le graphe complet, en permettant une interaction avec l'utilisateur sur ce graphe simplifié, enfin en reconstituant le graphe complet à partir du graphe simplifié.

SOUS-MODULE : Simplification du graphe

INPUT : Graphe complet dans la base de données locale des spécifications

OUTPUT : La matrice contenant les arêtes du graphe simplifié, le vecteur de ses sommets, l'ensemble des ponts et des pivots et l'ensemble des arêtes et sommets supprimés

TRAITEMENT : Extraction du graphe de la base de données locale des spécifications, réduction du graphe par suppression des objets isolés (ceux qui n'ont aucune arête), des objets pendants (ceux qui n'ont qu'une seule arête), des arêtes récursives (reliant deux fois le même sommet), des arêtes dédoublées (arêtes apparaissant deux fois) et des objets binaires (ceux desquels partent exactement deux arêtes), mémorisation de ces objets et arêtes et recherche des pivots et des ponts dans le graphe simplifié

SOUS-MODULE : Edition du graphe simplifié

INPUT : La matrice contenant les arêtes du graphe simplifié, le vecteur de ses sommets, l'ensemble des ponts et des pivots

OUTPUT : La matrice contenant la disposition du graphe simplifié à l'écran et la matrice contenant l'ensemble des coins formant les arêtes entre deux objets

TRAITEMENT : Dessine le graphe simplifié dans une fenêtre "MS-WINDOWS" suivant la méthode de la diagonale et permet à l'utilisateur de modifier la disposition des sommets. Les arêtes suivent ces sommets automatiquement

---

SOUS-MODULE : Reconstitution du graphe complet

INPUT : La matrice contenant la disposition du graphe simplifié à l'écran et la matrice contenant l'ensemble des coins formant les arêtes entre deux objets, le vecteur des sommets et l'ensemble des sommets et des arêtes supprimés

OUTPUT : La base de données locale graphique

TRAITEMENT : Réinsertion des sommets et des arêtes supprimés dans la matrice contenant les arêtes et dans le vecteur des sommets du graphe simplifié en conservant la disposition initiale des arêtes et sommets du graphe simplifié. Sauvegarde de cette matrice et de ce vecteur dans la base de données graphique.

NOM : ReadGraph (flag)

FONCTION : Constitue un vecteur des sommets et une matrice creuse des arêtes du graphe complet

INPUT : Le graphe complet dans la base de données locale des spécifications

flag : identifiant de la matrice creuse (INTEGER)

OUTPUT :La matrice creuse constituée des arêtes du graphe identifiée par flag

graph : vecteur des sommets du graphe

FORMAT : Base de données locale des spécifications : global, DBLOC

flag : paramètre, INTEGER

graph : vecteur global, VERT

TRAITEMENT : Création du vecteur de tous les sommets (identifiés par la DBKey des objets D.S.L.) du graphe complet

Création de la matrice creuse, spécifiant si oui ou non une arête existe entre deux sommets. L'élément (i, j) et l'élément (j, i) de la matrice sont positionnés à 1 s'il existe une arête entre l'objet mémorisé à la position i et l'objet mémorisé à la position j du vecteur

---

NOM : flag = SimpGraph (flag, pile)

FONCTION : Simplifie le graphe en appliquant la méthode décrite en 4.3.2.2.

INPUT : flag : identifiant de la matrice creuse (arêtes du graphe)

graph : vecteur des sommets du graphe

OUTPUT :flag : matrice creuse compressée après simplification

vecteur compressé des sommets restants après simplification

pile : liste des sommets et des arêtes supprimés

FORMAT : flag : paramètre, INTEGER

pile : paramètre, pointeur, PTRPILE

graph : vecteur global, VERT

TRAITEMENT : Supprime du graphe complet les objets isolés, les objets pendants, les arêtes récursives, les arêtes dédoublées et les objets binaires et les sauve dans une liste pour la future restauration

---

NOM : find_br_piv (flag, bridge, pivot)

FONCTION : recherche les ponts et les pivots dans le graphe simplifié

INPUT : flag : identifiant de la matrice creuse compressée

graph : vecteur compressé des sommets restants après simplification

OUTPUT :bridge : ensemble des arêtes qui sont des ponts

pivot : ensemble des sommets qui sont des pivots

FORMAT : flag : paramètre, INTEGER

graph : vecteur global, VERT

bridge : paramètre, vecteur, EDGE-SET

pivot : paramètre, vecteur, VERTICE-SET

TRAITEMENT : cherche les ponts et les pivots du graphe simplifié et les sauve dans deux vecteurs "bridge" et "pivot"

NOM : draw (flag, bridge, pivot, rep, coin, vertvect)

FONCTION : affichage du graphe simplifié

INPUT : flag : identifiant de la matrice compressée

graph : vecteur des sommets restants

bridge : ensemble des ponts

pivot : ensemble des pivots

OUTPUT :rep : matrice : représentation du graphe dans une grille

coin : matrice de pointeurs : position des coins des arêtes dans la grille

vertvect : position des objets dans la grille

FORMAT : flag : paramètre, INTEGER

graph : vecteur global, TVECTOR

bridge : paramètre, vecteur, TEDGE

pivot : paramètre, vecteur, TVERTICE

rep : paramètre, matrice, TREP

coin : paramètre, matrice (TCOIN) de pointeurs vers les coins (TLISTCOIN)

vertvect : paramètre, vecteur VERT

TRAITEMENT : En utilisant la méthode de la diagonale, crée une matrice contenant l'arrangement du graphe à l'écran : pour chaque cellule de la grille, définit l'objet contenu dans la cellule et le nombre de coins (avec une distinction pour leur orientation). Crée une matrice contenant le numéro de la cellule où se trouvent les coins d'une arête entre deux sommets

---

NOM : windraw (rep, coin, vertvect)

FONCTION : Calcule le déplacement dans une cellule pour chaque coin

INPUT : rep : matrice de la représentation du graphe dans la grille

coin : position des coins des arêtes dans la grille

vertvect : position des objets dans la grille

OUTPUT : coin : matrice mise à jour contenant la position des coins des arêtes dans la grille et leur déplacement relatif dans la cellule de la grille

FORMAT : rep : paramètre, matrice, TREP

coin : paramètre, matrice (TCOIN) de pointeurs vers les coins (TLISTCOIN)

vertvect : paramètre, vecteur, VERT

TRAITEMENT : Calcule le déplacement à l'intérieur de la cellule pour chaque coin afin éviter les croisements non nécessaires

---

NOM : WinMain

fonction principale de "MS-WINDOWS" : affiche le graphe simplifié dans une fenêtre et s'occupe des messages

---

NOM : MoveObject (window handle, X-coord, Y-coord)

FONCTION : Edition du graphe simplifié

INPUT : window handle : fenêtre (objet) dans laquelle l'utilisateur pousse sur le bouton de la souris

X-coord : déplacement en X de la souris dans la fenêtre

Y-coord : déplacement en Y de la souris dans la fenêtre

rep : matrice : représentation du graphe dans la grille

coin : position des coins des arêtes dans la grille

vertvect : position des objets dans la grille

OUTPUT :rep : matrice mise à jour : représentation du graphe dans la grille

coin : position mise à jour des coins des arêtes dans la grille

vertvect : position mise à jour des objets dans la grille

FORMAT : window handle : paramètre, HWND (type MS-WINDOWS prédéfini)

X-coord : paramètre, INTEGER

Y-coord : paramètre, INTEGER

rep : paramètre, matrice, TREP

coin : paramètre, matrice (TCOIN) of pointeurs vers les coins (TLISTCOIN)

vertvect : paramètre, vecteur, VERT

TRAITEMENT : Permet à l'utilisateur de modifier la position des objets en enfonçant le bouton de la souris sur un objet. Selon la position de la souris dans la fenêtre objet, celui-ci sera déplacé vers le haut, le bas, la gauche ou la droite (figure 5.2.). Pendant le déplacement d'un objet, toutes les arêtes reliées à cet objet le suivent, ce qui signifie création ou suppression des coins et suppression des "U" (figure 5.3.)

Remarque importante : Notons qu'à ce niveau, en ne déplaçant que les objets, il est impossible à l'utilisateur de supprimer un éventuel croisement.

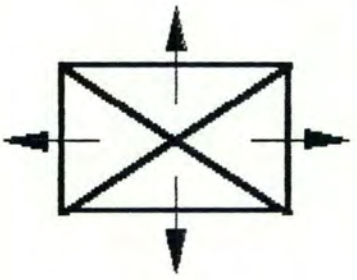


figure 5.2.

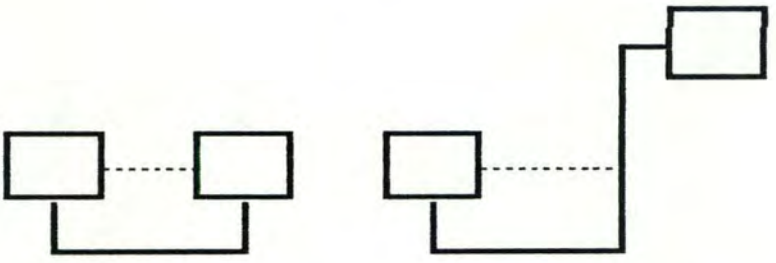


figure 5.3.

5.3.3. Format des données

La figure 5.4. présente les transformations de données à travers les différents sous-modules du générateur de brouillon. Les flèches peuvent être interprétées par "se transforme en".

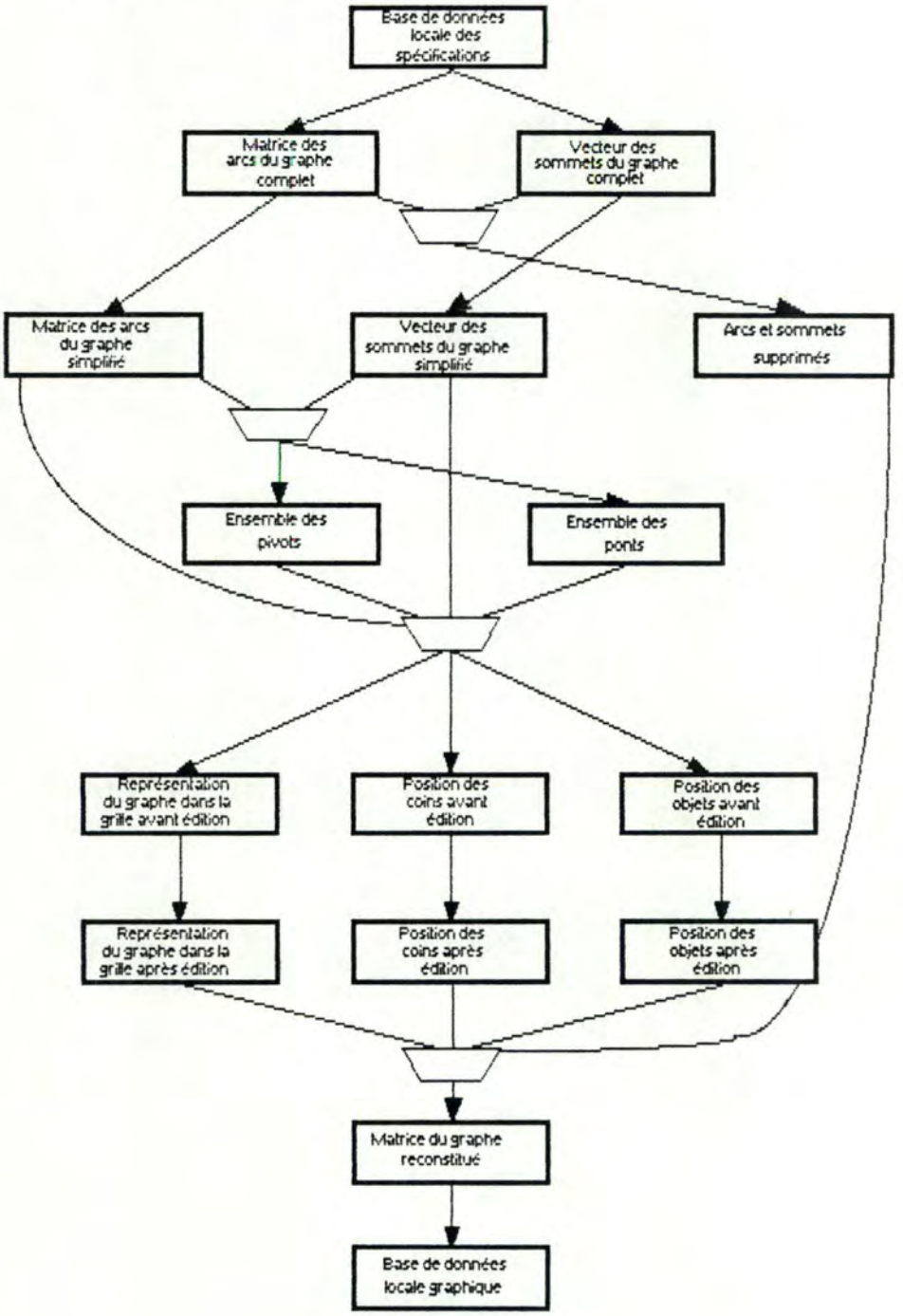


figure 5.4.

Remarque : les zones représentées sur fond noir sont réservées à un usage interne uniquement.

Base de données locale des spécifications

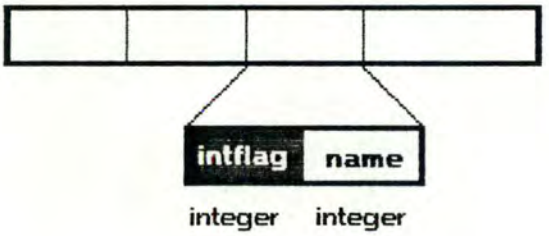
Elle est présentée en détail à la fin de ce chapitre et a été réalisée par l'équipe travaillant sur IDA.

Matrice des arêtes du graphe initial et simplifié

Il s'agit d'une matrice creuse, ce qui lui donne l'avantage d'une part, d'être dynamique (on ne réserve de la place en mémoire que pour les éléments non nuls), et d'autre part de jouir de fonctions de manipulation inexistantes pour une matrice classique, par exemple : suppression des lignes et des colonnes ne contenant aucune valeur significative (nécessaire pour la compression de la matrice), ajout d'une ligne ou d'une colonne (nécessaire pour la réinsertion de la pile des objets supprimés), ...

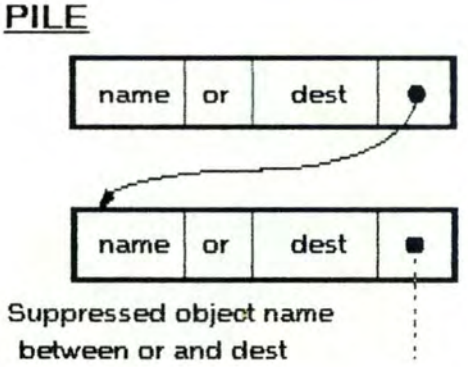
Vecteur des sommets du graphe initial et simplifié

VERT



Chaque élément du vecteur contient l'identifiant d'un objet (DBKey). Le vecteur est terminé par la valeur EOV (End Of Vector).

Ensemble des arêtes et sommets supprimés



Il s'agit d'une liste linéaire. Les objets et connexions supprimés sont mémorisée de la façon suivante :

suppression d'un objet isolé "O1" :

name = O1; or = NULL; dest = NULL;

suppression d'un objet pendant "O2" relié à "O3" :

name = O2; or = O3; dest = NULL;

suppression d'une arête récursive reliée à "O4" :

name = NULL; or = O4; dest = O4;

suppression d'une arête dédoublée entre "O5" et "O6" :

name = NULL; or = O5; dest = O6;

suppression d'un objet binaire "O7" entre "O8" et "O9" :

name = O7; or = O8; dest = O9;

Ensemble des pivots

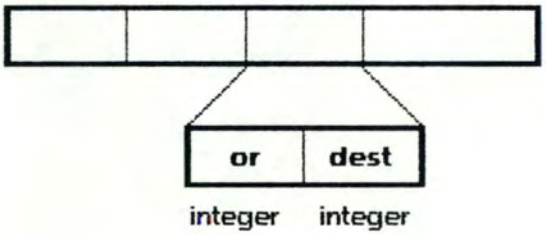
TVERTICE

		integer	

Chaque élément du vecteur contient l'identifiant d'un pivot (DBKey). Le vecteur est terminé par la valeur EOV.

Ensemble des ponts

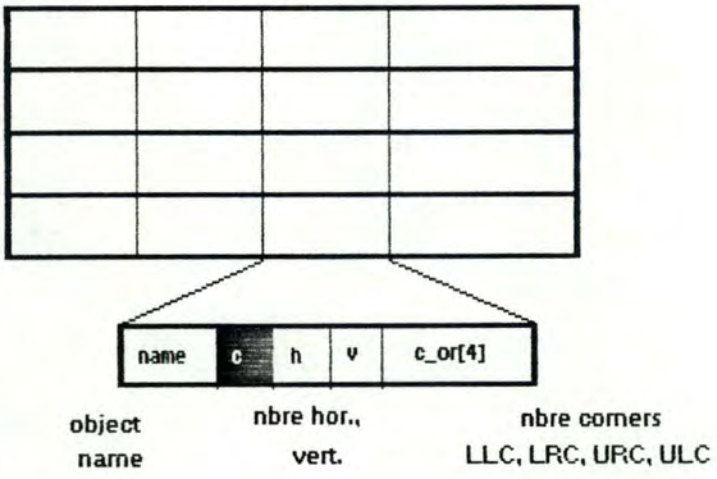
TEDGE



Chaque élément du vecteur contient les identifiants des objets formant un pont (DBKey). Le vecteur est terminé par la valeur EOV dans "or" et dans "dest".

Représentation du graphe simplifié dans la grille

REP TREP



La matrice REP a la même taille que la grille. Un élément de cette matrice correspond à une cellule de la grille. Chaque élément est composé de :

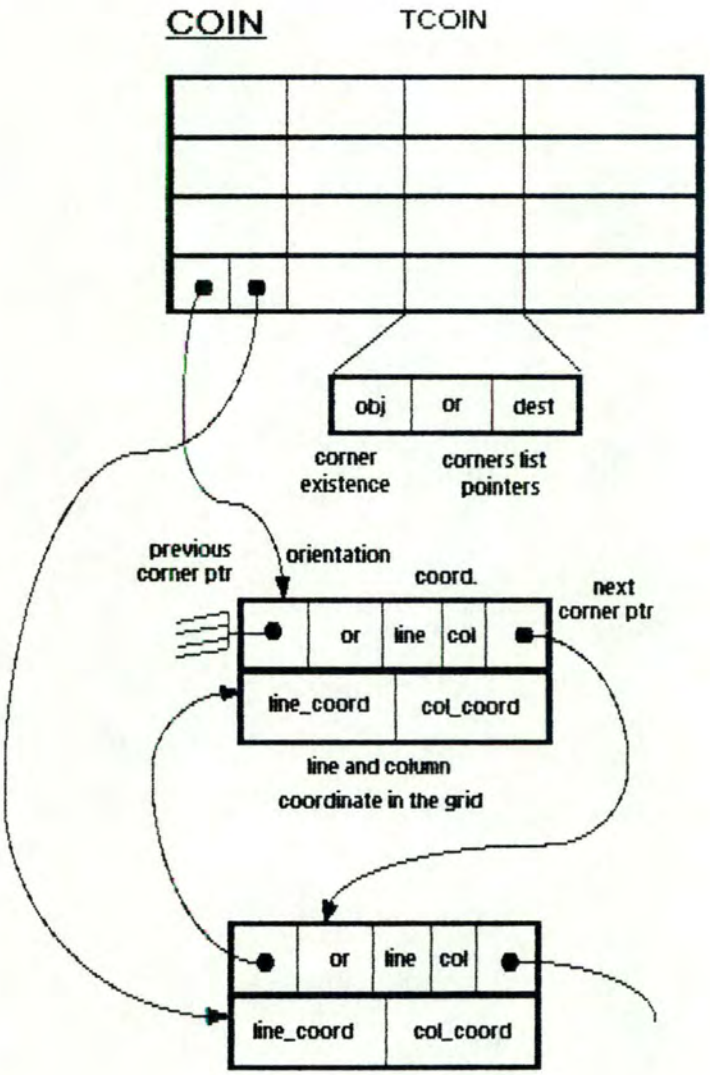
NAME : identifiant de l'objet se trouvant dans la case correspondante (NULL s'il n'y a pas d'objet).

H, V : nombre de lignes horizontales et verticales dans la case (toujours 0 s'il y a un objet).

C_OR : vecteur de quatre éléments : nombre de coins dans la case, en fonction de leur orientation (coin supérieur gauche (ULC), inférieur gauche (LLC), supérieur droit (URC) ou inférieur droit (LRC)).

Les éléments de ce vecteur sont tous à 0 s'il y a un objet dans la case.

Position des coins dans la grille



La matrice COIN a autant de lignes et de colonnes qu'il y a d'objets dans le graphe simplifié. Chaque élément de la matrice est composé d'un indicateur spécifiant si un coin existe ou non et de deux pointeurs vers des éléments décrivants chacun un coin de l'arête entre l'objet se trouvant en $i^{ème}$ position dans le vecteur des sommets et celui se trouvant en $j^{ème}$ position (pour l'élément (i, j) de la matrice). Le premier pointeur (OR)

indique l'adresse de l'élément correspondant au premier coin de l'arête (par rapport à i), le second pointe vers le dernier coin.

Tous les éléments représentant ces coins sont reliés entre eux (liste Bi-directionnelle). Ils sont composés de :

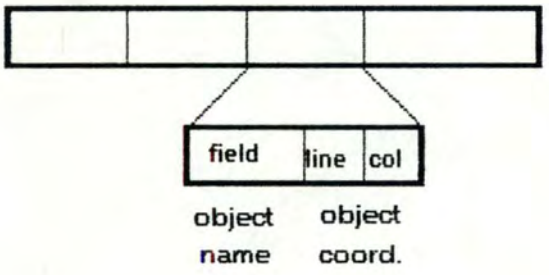
OR : orientation du coin.

LINE, COL : coordonnée du coin dans la grille.

LINE_COORD, COL_COORD : déplacement relatif du coin dans la cellule de cette grille.

Position des objets dans la grille

VERTVECT TVECTOR



5.3.4. Spécifications internes

FONCTION : ReadGraph (flag)

REALISATION : Parcours de la base de données des spécifications. Chaque objet rencontré est mémorisé en sauvegardant sa DBKey dans un vecteur. Chaque relation entre deux objets est mémorisée dans une matrice creuse. S'il existe une relation entre des objets aux positions i et j du vecteur des sommets, les éléments (i, j) et (j, i) de la matrice creuse sont positionnés à 1. Après le dernier élément du vecteur, la valeur EOV (end of vector) doit être positionnée.

FONCTION : flag = SimpGraph (flag, pile)

REALISATION : * Détection des sommets isolés (ceux qui n'ont aucune arête) et mémorisation de ceux-ci dans la pile des objets et connexions supprimés

* *Tant que la simplification n'est pas terminée :*

détection des sommets pendants (ceux qui n'ont qu'une arête) et mémorisation de ceux-ci dans la pile;

détection des arêtes récursives (reliant deux fois le même sommet) et mémorisation de celles-ci dans la pile;

détection des sommets binaires (ceux qui ont exactement deux arêtes) et mémorisation de ceux-ci dans la pile. Si la suppression d'un sommet binaire crée une arête dédoublée, elle est immédiatement supprimée et mémorisée dans la pile;

* Compression de la matrice creuse (suppression des lignes et colonnes correspondant à un objet supprimé) et compression du vecteur des sommets.

FONCTION : find_br_piv (flag, bridge, pivot)

REALISATION : * Construction du spanning-tree (arbre contenant tous les sommets du graphe simplifié et une partie de ses arêtes). Cet arbre est construit en profondeur d'abord.

* Pour chaque arête du graphe n'appartenant pas au spanning-tree, détermination d'un circuit composé de cette arête et d'arêtes du spanning-tree et numérotation de ce circuit de la façon suivante :

- Si aucune arête du circuit n'est marquée, on les marque toutes avec une nouvelle valeur (valeur supérieure à toutes les valeurs ayant déjà servi pour marquer les autres circuits).

- Si certaines arêtes du circuit sont déjà marquées, toutes avec la une même valeur, on marque les autres arêtes de ce circuit avec cette valeur.

- Si certaines arêtes du circuit sont marquées avec des valeurs différentes, on marque toutes les arêtes de ce circuit avec la plus petite de ces valeurs et on marque toutes les arêtes ayant une de ces différentes valeurs avec la plus petite d'entre elles.

* Les pivots sont les sommets reliés à des arêtes numérotées avec des valeurs différentes.

* Les ponts sont les arêtes non numérotées.

FONCTION : draw (flag, bridge, pivot, rep, coin, vertvect)

REALISATION : Mise en page du graphe simplifié. Les sommets et les arêtes sont positionnés dans une grille grâce à la méthode de la diagonale, ce qui signifie que les ponts et les pivots sont placés en premier lieu. Un pont est une arête entre deux sommets qui sont des pivots. Les deux sommets composant un pont sont disposés sur une même horizontale. Les autres sommets sont ensuite placés autour des ponts et des pivots suivant une diagonale pour chaque sous-graphe connexe du graphe. Après avoir placé les sommets, les chemins utilisés par les arêtes sont calculés comme suit :

* Toutes les arêtes ont toujours un et un seul coin
* Les objets consécutifs sur l'écran sont reliés en premier lieu
* Pour chaque arête non encore placée, la position du coin peut être sur la même ligne que l'objet le plus haut ou sur la même ligne que l'objet le plus bas. Le critère de sélection est, par ordre d'importance :
  - le chemin qui provoque le moins de croisements
  - le chemin qui traverse les cellules les moins occupées
  - le chemin dont la première et la dernière cellule sont les moins occupées
  - le chemin qui part horizontalement de l'objet placé le plus haut

Cette disposition est mémorisée dans deux matrices. La première est la représentation de la grille : pour chaque cellule, mémorisation soit de l'objet contenu dans cette cellule, soit du nombre de lignes horizontales et verticales et du nombre de coins avec une distinction de leur orientation

contenus dans cette cellule. L'autre matrice contient la position dans la grille des pliures des arêtes entre deux sommets donnés.

FONCTION : windraw (rep, coin, vertvect)

REALISATION : Calcul du déplacement de chaque coin dans une cellule de la grille de manière à éviter les croisements non nécessaires et de manière à ce que les arêtes soient judicieusement réparties. Ce calcul se fait en utilisant la matrice "rep" qui contient le nombre de lignes et de coins inclus dans cette cellule et dans les cellules qui sont traversées par l'arête contenant ce coin.

FONCTION : WinMain ( )

REALISATION : Selon la philosophie de "MS-WINDOWS", dessine le graphe sur une page graphique et renvoi les différents messages à la fonction correspondante. Chaque objet est représenté par une fenêtre fille et les arêtes sont dessinées dans la fenêtre mère.

FONCTION : MoveObject (window handle, X-coord, Y-coord)

REALISATION : Le mode "Edit" permet de déplacer un objet (les objets sont représentés par des rectangles ayant quatres parties, délimitées par ses diagonales). Quand l'utilisateur enfonce le bouton de la souris dans le triangle du haut, l'objet monte, dans celui du bas, l'objet descend, dans celui de droite, l'objet va à droite et dans celui de gauche, l'objet va à gauche (figure 5.2.). Deux objets ne peuvent être dans une même cellule

de la grille, et aucun objet ne peut quitter l'écran. Un objet ne peut recouvrir aucune arête ou partie d'arête. Toutes les arêtes suivent les objets déplacés. Si un "U" est créé entre deux objets (figure 5.3.), la fonction le supprime et le remplace par une ligne horizontale ou verticale si les deux objets sont sur la même ligne, ou le réduit sinon. Les "U" sont supprimés ou réduits si les nouvelles lignes ne doivent pas traverser un objet ou créer un croisement. Les lignes ou les coins affectés par le déplacement d'un objet sont mémorisés dans une liste et à la fin de la fonction, leur déplacement dans la cellule de la grille est recalculé et ils sont redessinés. Seulement la partie nécessaire de l'écran doit être redessinée car cela prendrait trop de temps de redessiner l'écran tout entier. Après chaque mouvement, la matrice des coins et le vecteur des sommets sont mis à jour.

5.4. Implémentation

Le générateur de brouillon a été développé en langage C (MS-C). Pour gérer l'environnement multi-fenêtres, nous avons utilisé le "Development Toolkit" de MS-WINDOWS. Ce logiciel comprend un ensemble de librairies et il a la particularité de pouvoir aborder la programmation par objets. En effet, à chaque classe de fenêtre est associé une fonction de gestion des messages. Chaque classe de fenêtre peut donc être gérée de façon différente par le système, sans que le concepteur ne se soucie, par exemple, de la position de la fenêtre sur l'écran. Nous parlerons plus longuement de ce système dans l'annexe.

Le seul compilateur utilisable avec le Toolkit de MS-WINDOWS est le compilateur de Microsoft. Il a l'avantage de posséder un "debugger" symbolique de très bonne qualité, mais qui est inutilisable lors du développement avec MS-WINDOWS. Aussi, avons nous travaillé le plus longtemps possible en dehors de cet environnement de manière à pouvoir utiliser ce "debugger". Pour tout ce qui est de l'interaction avec l'utilisateur, malheureusement, nous avons été obligé d'introduire MS-WINDOWS. Le seul "debugging" alors possible peut être de deux types. Soit l'installation d'un second terminal sur lequel le système envoi un ensemble d'informations qui ne sont pas directement utiles (contenu de la pile, nom de fonction interne, ...). Soit l'envoi d'une trace à l'imprimante (même une simple trace à l'écran est inimaginable, l'affichage d'un message nécessitant la création d'une fenêtre, une interaction de l'utilisateur, en l'occurrence le concepteur, pour la faire disparaître et enfin un rafraîchissement de l'écran !).

Il faut savoir enfin que le "debugging" est une opération fastidieuse, spécialement avec MS-WINDOWS, le programme ne se déroulant pas séquentiellement (gestion d'une liste de messages provenant aussi bien de l'utilisateur que de fonctions internes au système qui ne sont pas toujours gérés dans l'ordre dans lequel ils arrivent).

Nous avons adopté comme principe de représenter les objets du graphe simplifié comme étant chacun une fenêtre fille. Cette solution a l'avantage de pouvoir traiter les actions de l'utilisateur d'une manière différente selon que son curseur est positionné sur un objet ou non. De plus, s'il se trouve sur un objet, MS-WINDOWS en détecte automatiquement l'occurrence.

Un autre problème important que nous avons rencontré est l'utilisation de structures lourdes et complexes pour gérer par exemple la représentation du graphe, la position des coins, ...

Nous avons donc développé un générateur de brouillon dédié au modèle E.R.A. d'IDA.

Pour commencer, nous avons un graphe qui est la représentation des entités et relations du modèle à reconstituer. Ce graphe devra être construit à partir de la base de données locales des spécifications extraite de la base de données centrale.

Nous simplifions ce graphe au maximum suivant l'algorithme présenté au chapitre 4 (4.4.2.4.).

Nous disposons ensuite le graphe simplifié suivant la méthode de la diagonale que nous avons exposée en 3.2.2.1. Nous présentons alors le diagramme à l'analyste qui peut le réorganiser à sa meilleure convenance. Nous avons limité l'édition au déplacement des objets. Les connexions suivent l'objet déplacé et se simplifient si nécessaire. Par exemple, si un "U" est créé entre deux objets, il est raccourci ou supprimé (nous présentons plus longuement les "U" dans les spécifications externes et internes de la fonction "MoveObject").

Il reste à implémenter la réinsertion des simplifications pour retrouver le graphe complet. Tous les objets supprimés du graphe simplifié sont mémorisés dans une pile. Il est indispensable de réinsérer les objets dans l'ordre **inverse** de leur suppression,

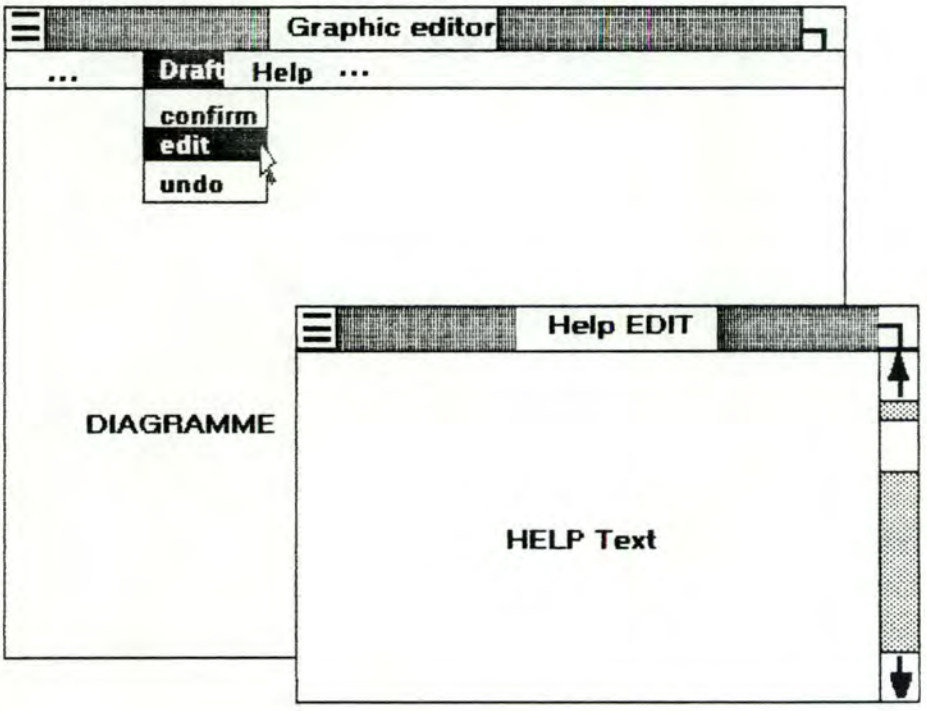
puisque certaines parties du graphe auront pu disparaître complètement lors de la simplification.

Le graphe ainsi reconstitué sera présenté à l'analyste qui aura la possibilité de l'éditer à nouveau. Le pilote du modèle E.R.A. sera responsable de cette édition.

Nous n'avons pas eu le temps d'implémenter la partie "HELP" de notre générateur de brouillon. Mais nous y avions songé et la fenêtre qui devrait le contenir existe déjà.

Vu les fonctionnalités de MS-WINDOWS, il est possible de concevoir un module "HELP" qui, à notre avis, serait tout à fait ergonomique.

Lorsque l'utilisateur demande le "HELP", une fenêtre apparait au centre de l'écran. Le texte affiché dans cette fenêtre explique le "HELP". Pour avoir une explication concernant n'importe quelle fonction du générateur, l'utilisateur doit sélectionner dans le menu de la fenêtre principale l'item à propos duquel il désire des renseignements. Et la fenêtre du "HELP" affiche immédiatement une page texte concernant la fonction choisie par l'utilisateur.



Fichier d'index (IDX)

Fichier de données (DBF)

Index des relations	
Code de relation	Clé de relation
Int	Pointer

Descripteur de relation				
Code de relation	Coins de la relation	Information graphique	Date de dernière mise à jour	Relation suivante
Int	Pointer * 4	Pointer * 4	(Long.Long)	Pointer

(Voir B.D. graphique)

Descripteur de texte				
Code de relation	Code d'objet maître	Code d'entrée texte	Date de dernière màj	Relation suivante
Int	Pointer	Pointer	(Long.Long)	Pointer

Entrée de texte	
Longueur	Texte
Int	Char * previous

Chaîne des relations	
Suivante	Clé de relation
Pointer	Pointer

Index des objets	
Nom d'objet	Clé d'objet
Char * 31	Pointer

Descripteur d'objet				
Nom d'objet	Code d'objet	Dernière mise à jour	Clé de synonyme	Liste des relations
Char * 31	(Int.Int)	(Long.Long)	Pointer	Pointer

Index des synonymes		
Nom de synonyme	Clé de synonyme	Clé de maître
Char * 31	Pointer	Pointer

Descripteur de synonyme	
Nom de synonyme	Synonyme suivant
Char * 31	Pointer

Conclusion

Nous avons essayé dans ce mémoire de définir une méthode pour aborder un éditeur graphique dédié à un langage de spécification.

Nous résumons maintenant le cheminement que nous avons suivi et rappelons les étapes importantes de l'élaboration d'un éditeur graphique dédié au langage de spécification D.S.L.

Pour commencer, nous avons analysé les différentes possibilités offertes par le traitement graphique. Nous n'avons relevé que les plus usitées. Nous avons essayé de mieux comprendre la psychologie de l'utilisateur, car il nous paraissait important de connaître son comportement face à un éditeur graphique, quel qu'il soit. Nous avons ensuite tenté de définir une méthodologie de travail de l'analyste pour y découvrir un scénario d'utilisation d'un éditeur graphique dédié au langage de spécification qu'il utilise.

Nous avons alors analysé deux systèmes de saisie existants : Structured Architect et I.E.W. (Information Engineering Workbench). Sur base de nos résultats nous avons pu concilier nos idées et celles développées dans ces deux éditeurs. Dès lors, nous avons pu aborder l'architecture d'un éditeur graphique en saisie dédié à IDA. Nous avons opté pour un compromis entre le texte et les graphiques, étant donné le contenu informationnel important des différents aspects du langage D.S.L. Nous avons pris

l'option également de pouvoir piloter l'analyste lors de la conception de son schéma en effectuant les contrôles syntaxiques et sémantiques au fur et à mesure de son évolution.

Pendant notre stage à l'Université du Michigan, nous avons développé le DES (Display Edit Server) qui est un éditeur de saisie graphique indépendant de toute méthodologie. Ce système est beaucoup plus large que l'éditeur dédié à IDA. Ici, aucun pilotage n'est possible puisque la méthodologie utilisée est inconnue au moment de la saisie. Les contrôles syntaxiques et sémantiques se feront donc via une sorte de compilateur instancié à la méthodologie utilisée.

Nous avons donc étudié de plus près les deux utilisations possibles d'un éditeur graphique : l'éditeur face auquel nous pensons et l'éditeur face auquel nous dessinons.

Enfin, nous avons abordé l'éditeur de restitution graphique. Ici encore, nous avons analysé deux systèmes existants : celui développé par le Professeur Batini et I.E.W. Nous nous sommes également inspirés des résultats pour développer l'architecture d'un éditeur graphique de restitution dédié à IDA et plus particulièrement au modèle E.R.A.

En ce qui concerne la conception de l'éditeur proprement dite, nous avions deux solutions intermédiaires qui comportaient chacune un gros inconvénient. La première, manuelle, ne permettait pas une vue globale du diagramme. La deuxième, nécessitant la détection de la planarité d'un graphe, était trop coûteuse en temps de calcul et aucun algorithme ne nous permettait d'affirmer qu'un graphe était planaire. Nous avons donc dû les abandonner. Nous avons cependant conservé l'idée de simplification. Celle-ci avait été développée pour détecter la planarité d'un graphe. Nous pouvions réutiliser le graphe simplifié pour le proposer à l'utilisateur comme brouillon de son modèle. Ce brouillon est présenté suivant la méthode de la diagonale, inspirée par la méthode développée chez Arthur Young Proware pour I.E.W. L'utilisateur est libre d'en modifier la représentation. Nous reconstruisons alors le graphe complet en réinsérant les simplifications effectuées.

Nous terminons la présentation du générateur de brouillon en présentant les spécifications des modules les plus importants.

Le développement de logiciels utilisant MS-WINDOWS est une tâche complexe puisque l'exécution de tels programmes est non séquentielle et donc difficilement contrôlable. La philosophie de MS-WINDOWS est la programmation par événements, chacun de ceux-ci correspondant à un message. De plus, les messages peuvent être générés soit par une opération d'entrée-sortie de l'utilisateur soit par l'application elle-même. La non séquentialité provient des priorités différentes de chacun des messages.

Compte tenu de cette extrême complexité, il nous a semblé intéressant de développer en annexe, une illustration des fonctionnalités et des concepts fondamentaux que nous avons utilisés pour nos deux applications.

Le lecteur intéressé trouvera dans le second tome les programmes sources du générateur de brouillon. Il y trouvera également les spécifications propres à chacune des fonctions.

Nous avons développé un éditeur graphique de restitution inséré dans le projet IDA. La similitude des architectures de saisie et de restitution nous permet de penser qu'une extension probable de notre travail serait un éditeur de saisie. En effet, dans une étape ultérieure, une fois la tâche de l'éditeur de restitution accomplie, l'utilisateur pourra modifier le contenu sémantique du diagramme, ce qui entraînera une mise à jour de la base de données des spécifications. Si cette opération est effectuée à partir d'un schéma non existant, nous n'obtiendrons rien d'autre qu'un éditeur graphique de saisie.

La méthode que nous avons proposée est une approche heuristique. Le développement d'un éditeur graphique est un travail complexe étant donné le nombre d'approches possibles. En effet, nous avons du choisir une solution parmi toutes celles qui s'offraient à nous. De plus, nous ne pouvions pas concevoir un logiciel non

performant. Nous devions aussi éviter de rejeter la complexité du développement du côté de l'utilisateur en lui proposant un système non convivial.

Dès que l'équipe IDA disposera de cet éditeur, il serait intéressant d'approfondir nos préoccupations exposées dans le second chapitre, à savoir l'étude du comportement de l'analyste face à ce logiciel, ainsi que d'effectuer une analyse comparative avec les autres outils graphiques qui existeront.

Signalons que le traitement graphique n'en est qu'à ses premiers pas et que les logiciels devront être constamment révisés suite aux améliorations effectuées en ce domaine.

Il y a quelques années, lors de l'apparition de la gestion par menus, de nombreuses personnes croyaient être en présence d'un système idéal. Or, à l'usage, elles se sont aperçues que ce n'était pas le cas. Aujourd'hui, l'apparition du traitement graphique suscite les mêmes réactions. Il est fort probable que, d'ici quelques temps, l'utilisation du traitement graphique sera très différente suite aux études qui seront effectuées concernant la méthodologie d'utilisation de celui-ci.

Bibliographie

[BATI83] C. Batani, M. Talamo, R. Tamassia : An Algorithm for Automatic Layout of Entity Relationship Diagrams, in Entity-Relationship Approach to Software Engineering, 1983

[BATI85] C. Batani, E. Nardelli, M. Talamo, R. Tamassia : GINCOD : A Graphical Tool for Conceptual Design of Data Base Application, in Computer Aided atabase Design, 1985

[BODA83] François BODART, Yves PIGNEUR : Conception assistée des applications informatiques. 1. Etude d'opportunité et Analyse conceptuelle, Masson, 1983

[CLAN83] Chuck CLANTON : The Future of Metaphor in Man-Computer Systems, in Byte, December 1983

[DEO74] Narsingh DEO : Graph Theory with Applications to Engineering and Computer Science, Prentice-Hall Inc, 1974

[DVOR86] DVORAK : The "WYSIWYG" Mania, in Computer, 1986

[HAYE81] Phil HAYES, Eugène BALL, Raj REDDY : Breaking the Man-Machine Communication Barrier, in IEEE, Computer, March 1981.

[IEWa] Arthur Young Proware : Information Engineering Workbench, presentation paper, 1987

[IEWb] Arthur Young Conseil : Information Engineering Workbench : La Nouvelle Révolution Informatique

[JOHN72] David E. JOHNSON, Johnny R. JOHNSON : Graph Theory With Engineering Applications, The Ronald Press Company, New York, 1972

[LEFE87] Jacques LEFEVRE, Benoit SACRE : Interprétation graphique de mesures de la performance d'un système d'information et production automatique du schéma de la dynamique, Travail de fin d'études, FNDP, 1987.

[LIEB85] Henry LIEBERMAN : There's More to Menu Systems Than Meets The Screen, in ACM, Computer Graphics, August 1985.

[MARC78] T. de MARCO : Structured Analysis and System Specification, YOURDON, 1978.

[MEAD86] Jon MEADS : Computer Graphics Standards, in ACM, Computer Graphics, May 1986.

[MEYR82] Norman MEYROWITZ, Andries VAN DAM : Interactive Editing Systems : Part 1, in Computing Survey, September 1982

[MORA81] T.P. MORA : An Applied Psychology of the User, in Computing Survey, March 1981.

[PRIS86a] Program for Research in Information Systems Engineering : Concepts Document for the Display Edit Server, November 1986.

[PRIS86b] Program for Research in Information Systems Engineering : Conceptual Overview of SEEOF, September 1986.

[READ85] Georg RAEDER : A Survey of Current Graphical Programming Techniques, in IEEE, Computer, August 1985.

[ROY69] Bernard ROY : Algèbre moderne et théorie des graphes, Tome 1, DUNOD, Paris, 1969.

[S.A.86] ISDOS Inc. : Structured Architect User's Guide, version 1.1., March 86, Ann Arbor, Michigan.

[SING83] Baldev SINGH, John C. BEATTY, Rhonda RYMAN : A Graphics Editor for Benesh Movement Notation, in ACM, Computer Graphics, July 1983.

[WARF83] Robert W. WARFIELD : The New Interface Technology - An Introduction to Windows and Mice, in Byte, December 1983.

[WRIG86] Thomas WRIGHT : Graphics Standard - A Personal Assesment, in ACM, Computer Graphics, May 1986.

Annexe : Illustration des fonctionnalités de MS-WINDOWS à partir de l'implémentation

A.1. Introduction

Le toolkit de MS-WINDOWS développé par Microsoft Corporation est un ensemble de librairies permettant de créer un environnement multi-fenêtres à partir de différents langages de programmation (C, Pascal, ...).

Le but de cette annexe est de décrire de façon simple les fonctionnalités de ce logiciel et sa philosophie. Nous l'illustrerons par des exemples extraits des programmes que nous avons réalisés. Nous ne visons bien sûr pas l'exhaustivité, un mémoire entier n'y suffirait pas. Nous espérons seulement pouvoir aider un programmeur à "entrer" dans la philosophie de MS_WINDOWS, de manière à lui faciliter la tâche lorsqu'il devra consulter le Programmer's Reference Manual lors de la conception de sa propre application.

Successivement, nous aborderons les points suivants :

Nous définirons une application au sens MS-WINDOWS, nous exposerons les fonctions que doit contenir une application. Puis nous illustrerons les différents types de fenêtres, les différents types de messages, comment créer des menus, des icônes, des fenêtres de dialogue, ... Enfin, nous aborderons la partie purement graphique du logiciel.

Signalons encore que c'est volontairement que nous n'avons pas traduit les termes spécifiques à MS-WINDOWS de façon à familiariser le lecteur avec ce vocabulaire utilisé tout au long des manuels et repris dans le nom de la plupart des fonctions.

A.2. Définition d'une application

Une application est tout programme utilisant les conventions de programmation pour accéder au système MS-WINDOWS et utilisant les fonctions de gestion de fenêtres (Window Functions). Une application doit contenir au moins une fonction principale toujours appelée WinMain et une ou plusieurs fonctions de gestion de fenêtres.

Le générateur de brouillon que nous avons développé est une application.

Notons qu'une application peut éventuellement, si elle ne nécessite aucune interaction avec l'utilisateur, ne pas utiliser de fenêtre. Une telle application sera appelée "application automate".

A.3. La fonction WinMain

WinMain est la fonction qui est appelée par le système d'exploitation quand l'application est lancée. Si l'application est lancée plusieurs fois (plusieurs occurrences), la fonction doit réagir de façon différente, car pour être plus économique en espace mémoire, MS-WINDOWS peut ne charger qu'une seule fois le code et celui-ci est partagé. Seules les données se trouvent plusieurs fois en mémoire. Nous verrons comment réagit la fonction dans l'exemple qui suit.

Tous les événements pouvant survenir lors de l'exécution de l'application sont gérés en terme de messages (exemples : déplacement de la souris, action au clavier, demande de fermeture de la fenêtre, sélection d'une option dans un menu, rafraîchissement de l'écran, ...). Tous ces messages sont envoyés à la fonction principale. Ils sont, pour la plupart d'entre eux, mis dans une file d'attente et gérés dans un ordre dépendant de la priorité du message. Le rôle de la fonction principale est de répercuter les messages à la Window Function correspondante.

Illustration :

```
int PASCAL WinMain (hInstance, hPrevInstance, lpszCmdLine, cmdShow)

HANDLE hInstance, hPrevInstance;
```

HANDLE : Type MS-WINDOWS : "unsigned integer"
hInstance : identifiant de l'instance courante de l'application
hPrevInstance : identifiant de la dernière instance de l'application. Si l'instance courante est la première (on n'a pas encore lancé cette application), hPrevInstance = 0

```
LPSTR lpszCmdLine;
```

LPSTR : Type MS-WINDOWS : "Long Pointer to a STRing"
lpszCmdLine est une chaîne de caractères contenant une ligne de commande (les paramètres du programmes). Elle peut être initialisée par la commande RUN du MSDOS. (exemple WRITE DOCUMENT.DOC)

```
int cmdShow;
```

cmdShow est un entier spécifiant comment la fenêtre doit être initialisée (en icône, en mode zoom, normale, ...). C'est le paramètre qui est passé à la fonction ShowWindow.

```
{
    MSG  msg;
```

MSG : Structure MS-WINDOWS contenant l'identifiant de la fenêtre qui doit recevoir le message, le type de message et une série de paramètres (cette structure sera présentés en A.5.)

```
    HWND hWnd;
```

HWND : Type MS-WINDOWS : identifiant pour une fenêtre

```
    HMENU hMenu;
```

HMENU : Type MS-WINDOWS : identifiant pour un menu

```
    RECT rect;
```

RECT : structure MS-WINDOWS : coordonnées d'un rectangle :
coin supérieur gauche : rect.top, rect.left
coin inférieur droit : rect.bottom, rect.right

```
    int  i, j;
    char name[2];
```

```
if (!hPrevInstance)
```

S'il n'existe pas encore d'instance (la fenêtre est créée pour la première fois)

```
{
    if (!GraphInit (hInstance))
```

Fonction d'initialisation : lecture des ressources (elles seront expliquées en A.6.) et initialisation de la classe des fenêtres (voir A.4.)

```
        return FALSE;
}
else
{
```

Une instance existe déjà : on accède à la zone des données d'une occurrence antérieure.

```
    GetInstanceData (hPrevInstance, (PSTR)szAppName, 10);
    GetInstanceData (hPrevInstance, (PSTR)szAbout, 10);
    GetInstanceData (hPrevInstance, (PSTR)szWindowTitle, 21);
    GetInstanceData (hPrevInstance, (PSTR)&hAccelTable,
                    sizeof(hAccelTable));
}
```

Création de la Tiled Window (voir A.4.).

```
hWnd = CreateWindow ((LPSTR)szAppName,
                    (LPSTR)szWindowTitle,
                    WS_TILEDWINDOW,
                    0,              /* x - ignored for Tiled Windows */
                    0,              /* y - ignored for Tiled Windows */
                    0,              /* cx - ignored for Tiled Windows */
                    0,              /* cy - ignored for Tiled Windows */
                    (HWND)NULL,     /* no parent */
```

```
    (HMENU)NULL,          /* use class menu */
    (HANDLE)hInstance,    /* handle to window instance */
    (LPSTR)NULL           /* no params to pass on */
);
```

- *Ensemble de fonctions à exécuter après la création de la fenêtre,*
- mais avant la gestion des messages

```
while (GetMessage ((LPMSG)&msg, NULL, 0, 0))
```

La fonction GetMessage place dans la structure msg le message le plus ancien (ou le plus prioritaire) se trouvant dans la queue, si un message existe. Dans le cas contraire, elle attend de manière passive, **c'est-à-dire qu'elle rend la main au système** ce qui permet aux autres applications d'obtenir le contrôle afin d'exécuter leurs messages.

```
{
    if (TranslateAccelerator (hWnd, hAccelTable, (LPMSG)&msg) == 0)
```

Transforme les accélérateurs qui sont des entrées au clavier en des actions dans les menus. Les accélérateurs seront expliqués au point A.6.

```
    {
        TranslateMessage ((LPMSG)&msg);
```

Quand l'utilisateur enfonce une touche au clavier, MS_WINDOWS reçoit une virtual key. Cette fonction a comme rôle de traduire ce codage dans la valeur du caractère de la touche enfoncée.

```
        DispatchMessage ((LPMSG)&msg);
```

Envoie le message en cours de traitement à la Window Function correspondant à la fenêtre à laquelle est adressé le message.

```
    }
}

return (int) msg.wParam;
```

}

A.4. *Les fenêtres*

Les fenêtres sont les éléments de base du système. Une fenêtre possède les mêmes fonctionnalités qu'un terminal à la différence près qu'elle ne monopolise pas les ressources hardware. C'est cette différence fondamentale qui permet à MS-WINDOWS de travailler en mode multi-fenêtres.

Une fenêtre est composée de différents éléments, la plupart d'entre eux étant optionnels :

- La Client Area est la zone dans laquelle apparait le texte et les graphiques affichés par l'application.
- La Caption Bar est la zone supérieure de la fenêtre. Elle contient une chaîne de caractères où peut apparaître le nom de l'application, l'action en cours (par exemple "Loading ..."), ou n'importe quel autre message. Elle peut également contenir une System Box (le menu système) et une Size Box (permettant de changer la taille de la fenêtre).
- Les menus
- Les Scroll Bars permettant le défilement de la Client Area
- Enfin, à chaque fenêtre de type Tiled Window peut être associée une icône, Bitmap servant de représentation de la fenêtre quand elle est mise en attente.

L'apparence exacte d'une fenêtre dépend de la définition de sa classe et des paramètres spécifiés lors de sa création. A chaque fenêtre doit être associée une classe, et une classe peut servir pour un nombre illimité de fenêtres. Toutes les fenêtres ayant la même classe réagiront de la même façon aux messages.

Illustration : création d'une classe de fenêtre

```
pGraphClass     = (PWNDCLASS) LocalAlloc (LPTR, sizeof(WNDCLASS));

  pGraphClass->hCursor          = LoadCursor (NULL, IDC_ARROW);
```

Les fenêtres de la classe "graph" auront comme curseur la flèche inclinée (curseur prédéfini)

```
  pGraphClass->hIcon            = LoadIcon (hInstance, (LPSTR)szAppName);
```

Elles auront comme icône le Bitmap se trouvant dans le fichier spécifié derrière le mot clé ICON, associé à la chaîne de caractères contenus dans szAppName ("graph"). L'icône est spécifiée dans le fichier des ressources (graph ICON graph.ico). Ce fichier sera expliqué dans le point A.6.

```
  pGraphClass->lpszMenuName = (LPSTR)szAppName;
```

Elles auront comme menus les définitions spécifié derrière le mot clé MENU, associé à la chaîne de caractères contenus dans szAppName ("graph") dans le fichier des ressources (graph MENU). Ce fichier sera expliqué dans le point A.6.

```
  pGraphClass->hInstance        = hInstance;
```

Identification de l'application enregistrant la classe dans le système.

```
  pGraphClass->lpszClassName = (LPSTR)szAppName;
```

Elles auront comme identifiant de classe la chaîne de caractères contenue dans szAppName ("graph")

```
  pGraphClass->hbrBackground = NULL;
```

Elles auront comme couleur de fond (background) la couleur spécifiée en paramètre (la liste des couleurs système se trouve dans le manuel de référence p. 256). Dans

ce cas ci, NULL signifie que la fonction de rafraîchissement de la fenêtre doit prendre le background en charge.

```
pGraphClass->style          = CS_HREDRAW | CS_VREDRAW;
```

Le style spécifié pour cette classe signifie qu'elles seront entièrement redessinées si leur taille change, aussi bien horizontalement que verticalement.

```
pGraphClass->lpfnWndProc    = GraphWndProc;
```

GraphWndProc est le nom de la Window Function associée à cette classe.

Il existe trois types de fenêtres : les Tiled Windows, les Child Windows et les Popup Windows.

La Tiled Window est la fenêtre principale d'une application. Sauf dans le cas où l'utilisateur effectue un zoom sur la fenêtre, les Tiled Windows ne peuvent se recouvrir l'une l'autre, et le système s'arrangera toujours pour qu'elles remplissent l'entièreté de l'écran. La fenêtre est rendue visible à l'écran par la fonction ShowWindow.

Illustration de création d'une Tiled Window :

```
hWnd = CreateWindow ((LPSTR)szAppName,
```

Classe de la fenêtre à créer (dans cet exemple : "graph")

```
                    (LPSTR)szWindowTitle,
```

Chaîne de caractères à placer dans la Caption Bar

WS_TILEDWINDOW,

Apparence de la fenêtre :
WS_TILEDWINDOW est équivalent à la combinaison de WS_TILED, WS_BORDER, WS_SYSMENU, WS_SIZEBOX et WS_CAPTION : création d'une Tiled Window possédant un cadre, un menu système, une Size Box et une Caption Bar.

0, ignoré pour une Tiled Windows (toujours 0)
0, ignoré pour une Tiled Windows (toujours 0)
0, ignoré pour une Tiled Windows (toujours 0)
0, ignoré pour une Tiled Windows (toujours 0)

(HWND)NULL,

Identifiant de la Parent Window (inexistant dans le cas d'une Tiled Window

(HMENU)NULL,

La fenêtre possède les menus spécifiés dans l'enregistrement de sa classe.

(HANDLE)hInstance,

Identifiant de l'instance associé à la fenêtre (passé par l'application à WinMain).

(LPSTR)NULL

Pointeur vers un "long integer" : paramètre éventuel à passer au message WM_CREATE. Dans cet exemple, il n'y a pas de paramètre.

);

Une Child Window est une fenêtre qui est affichée dans la Client Area d'une autre fenêtre. Elle est généralement utilisée pour diviser la Client Area en plusieurs partie ayant des fonctionnalités différentes. Une Child Window dépend toujours d'une autre fenêtre, sa Parent Window.

Illustration de création d'une Child Window :

```
Child[i].hWnd = CreateWindow ((LPSTR) szObject,
```

Classe de la fenêtre à créer (dans cet exemple : "object")

```
(LPSTR) NULL,
```

Pas de Caption Bar

```
WS_CHILDWINDOW | WS_VISIBLE,
```

Apparence de la fenêtre : création d'une Child Window initialement visible.

```
GridX * vertvect[i].col + 3,
```

Coordonnée en X de la Child Window dans la Client Area de sa Parent Window.

```
GridY * vertvect[i].line + 3,
```

Coordonnée en Y de la Child Window dans la Client Area de sa Parent Window.

GridX - 6,

Taille de la fenêtre en X.

GridY - 6,

Taille de la fenêtre en Y.

(HWND) hWnd,

Identifiant de la Parent Window

(HMENU) NULL,

La fenêtre possède les menus spécifiés dans l'enregistrement de sa classe.

(HANDLE) hInst,

Identifiant de l'instance associé à la fenêtre (passé par l'application à WinMain).

(LPSTR) NULL

Pointeur vers un "long integer" : paramètre éventuel à passer au message WM_CREATE. Dans cet exemple, il n'y a pas de paramètre.

);

Une Popup Window est une fenêtre qui peut être affichée n'importe où sur l'écran. Contrairement aux Child Windows, une Popup Window reste toujours entièrement visible (elle est toujours placée au-dessus des autres). C'est la raison pour laquelle elle est généralement utilisée comme fenêtre temporaire. Comme la Child Window, elle dépend

aussi d'une autre fenêtre, mais sa position à l'écran n'est pas limitée par les bornes de la Client Area de sa Parent Window. Notons que pour certaines applications particulières, une Popup Window peut être la fenêtre principale.

Remarquons qu'il existe une classe particulière de Popup Window prédéfinie : les Message Box. Une Message Box est une fenêtre contenant une Caption Bar, éventuellement une icône, une chaîne de caractères contenant le message à afficher et un Push Button (expliqué au paragraphe A.6.). Elle est crée par un simple appel à la fonction MessageBox.

Illustration de création d'une Popup Window :

```
hHelp = CreateWindow ((LPSTR)"Help",
```

Classe de la fenêtre à créer (dans cet exemple : "help")

```
(LPSTR)"HELP SCREEN",
```

Chaîne de caractères à placer dans la Caption Bar

```
WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU,
```

Apparence de la fenêtre : création d'une Popup Window initialement visible ayant une Caption Bar et un menu système.

```
pt.x + 50,
```

Coordonnée en X de la Popup Window dans la Client Area de sa Parent Window (ptx = coordonnées en x du coin supérieur gauche de la Client Area).

```
pt.y + 30,
```

Coordonnée en Y de la Popup Window dans la Client Area de sa Parent Window (pty = coordonnées en y du coin supérieur gauche de la Client Area).

```
3 * (GetDeviceCaps (hIC, HORZRES)) / 4,
```

Taille de la fenêtre en X (3/4 de la Parent Window).

```
3 * (GetDeviceCaps (hIC, VERTRES)) / 4,
```

Taille de la fenêtre en Y (3/4 de la Parent Window).

```
(HWND)hWnd,
```

Identifiant de la Parent Window

```
(HMENU)NULL,
```

La fenêtre possède les menus spécifiés dans l'enregistrement de sa classe (pas de menus ici).

```
(HANDLE)NULL,
```

Identifiant de l'instance associé à la fenêtre (passé par l'application à WinMain).

```
(LPSTR)NULL     /* no params to pass on */
```

Pointeur vers un "long integer" : paramètre éventuel à passer au message WM_CREATE. Dans cet exemple, il n'y a pas de paramètre.

```
);
```

A.5. Les messages et les Window Functions

Un message est l'interprétation de tout événement pouvant se produire lors de l'exécution d'une application. Si une application se déroule dans plusieurs fenêtres (le générateur de brouillon possède une Tiled Window, une Child Window par objet et, quand le Help est affiché, une Popup Window), le système redirige automatiquement le message vers la fenêtre concernée. C'est-à-dire que l'utilisateur enfonce le bouton de la souris dans une Child Window, le message sera envoyé à la Window Function de la classe de cette Child Window.

Les messages sont gérés, pour une fenêtre, par sa Window Function. En fait, il s'agit d'une grande alternative où, en fonction du message reçu, on effectue un ensemble d'actions particulières. Les messages sont généralement envoyés par le système, mais l'application peut elle-même s'envoyer un message ou en envoyer un à toute autre application (par les fonctions SendMessage ou PostMessage).

Un message est une structure ayant la forme suivante :

```
typedef struct {
  HWND hwnd;
        Identifiant de l'occurrence de la fenêtre qui a envoyé le message
  WORD message;
        Identifiant du message
  WORD wParam;
  LONG lParam;
        Paramètres, différents selon les types de messages
  DWORD time;
        Instant où le message a été envoyé
  POINT pt;
        Coordonnées de la souris au moment de l'envoi du message
} MSG;
```

Nous allons maintenant illustrer les quelques messages les plus utilisés en présentant une Window Function (celle de l'application développée aux Etats-Unis : le DES).

```
long FAR PASCAL DesWndProc (hWnd, message, wParam, lParam)
HWND hWnd;
unsigned message;
WORD wParam;
LONG lParam;

{
  switch (message)
  {
  case WM_SYSCOMMAND:
```

Message envoyé lors de la sélection d'une option dans le menu système. wParam contient l'identifiant de l'option sélectionnée.

```
    switch (wParam)
    {
    case IDSABOUT:
      DialogBox (hInst, MAKEINTRESOURCE(ABOUTBOX), hWnd, lpprocAbout);
```

Affichage de la Dialog Box About.

```
      break;
    default:
      return DefWindowProc (hWnd, message, wParam, lParam);
    }
    break;
```

case WM_ERASEBKGND:

Message envoyé quand une partie de la fenêtre doit être redessinée (par exemple quand on enlève une Popup Window) en préparation au rafraîchissement de la fenêtre.
wParam contient le Display Context (voir A.8.).

```
EraseBkgnd (hWnd, wParam);
break;
```

case WM_CLOSE :

Message envoyé quand une fenêtre va être fermée. C'est ici que l'application peut demander confirmation à l'utilisateur avant d'appeler la fonction DestroyWindow.

```
Close (hWnd);
break;
```

case WM_DESTROY:

Message envoyé après que DestroyWindow ait été invoqué, c'est-à-dire quand la fenêtre a disparu.
PostQuitMessage place un message particulier dans la file des messages pour signaler la fin de l'application.

```
PostQuitMessage (0);
break;
```

case WM_QUERYENDSESSION:

Message envoyé quand l'utilisateur a demandé la fin de la session à partir du MS_DOS. Si l'application ne veut pas se terminer, elle doit retourner la valeur "0".

```
return (EndSession (hWnd));
```

```
    break;

case WM_SIZE :
```

Message envoyé après que la taille de la fenêtre ait été modifiée ou que la fenêtre ait été déplacée (Child Window).
wParam contient le type de modification : mise en icône, zoom, ou autre.
lParam contient la nouvelle largeur (LOWORD(lParam)) et la nouvelle hauteur (HIWORD(lParam)) de la fenêtre.

```
    Size (hWnd);
    break;

case WM_PAINT:
```

Message envoyé quand une fenêtre doit être redessinée.
lParam contient un pointeur vers une structure contenant les informations nécessaires au rafraîchissement de l'écran (Display Context (décrit au point A.7.), coordonnées du rectangle à redessiner, ...).

```
    Paint (hWnd, lParam);
    break;

case WM_CREATE :
```

Message envoyé quand la fonction CreateWindow est appelée. WM_CREATE est effectué avant que la fenêtre ne soit rendue visible. C'est ici que l'application doit effectuer toutes les initialisations.
lParam contient un pointeur vers une structure contenant les paramètres de la fonction CreateWindow.

```
    Creation (hWnd);
    break;
```

```
case WM_COMMAND:
```

Message envoyé lors de la sélection d'une option dans un menu.
wParam contient l'identifiant de l'option sélectionnée.

```
  DesCommand (hWnd, wParam);
  break;

case WM_LBUTTONUP :
case WM_LBUTTONDOWN :
case WM_MOUSEMOVE :
```

Messages envoyés lors d'une action de la souris (ici, respectivement bouton gauche enfoncé, bouton gauche relâché et souris dépiacée).
wParam signale des informations au sujet de l'environnement au moment de l'envoi du message : l'autre bouton est enfoncé ou non, les touches "shift" et "control" sont enfoncées ou non.
lParam contient les coordonnées en x et en y de la souris au moment de l'envoi du message.
Remarque : dans le cas d'un double click, quatres messages sont envoyés : bouton enfoncé, bouton relâché, "double click" et bouton relâché.

```
  DesWndMouse (hWnd, message, wParam, MAKEPOINT (lParam));
  break;

case WM_HSCROLL :
  HScrolling (hWnd, wParam, LOWORD(lParam));
  break;

case WM_VSCROLL :
```

Messages envoyés lors d'une action sur une Scroll Bar, respectivement horizontale et verticale. Nous discuterons ce message dans la fonction VScrolling ci-dessous.
wParam contient le type d'action effectuée sur la Scroll Bar.

```
    VScrolling (hWnd, wParam, LOWORD(lParam));
    break;

  default:
```

Les autres messages ne nécessitent pas de traitement particulier dans cette application. Ils seront donc traités par défaut par MS_WINDOWS (fonction DefWindowProc).

```
    return DefWindowProc (hWnd, message, wParam, lParam);
    break;
  }
  return (0L);
}
```

```
void VScrolling (hWnd, wParam, newpos)
HWND hWnd;
WORD wParam;
int newpos;
```

newpos = LOWORD(lParam) : nouvelle position de l'élévateur dans le cas où l'utilisateur l'a déplacé (SB_THUMBPOSITION)

```
{
 switch (wParam)
  {
   case SB_LINEUP :
```

Défilement d'une ligne vers le haut ou vers la gauche.

```
     ...
     break;
```

case SB_PAGEUP :

Défilement d'une page vers le haut ou vers la gauche.

...
break;

case SB_LINEDOWN :

Défilement d'une ligne vers le bas ou vers la droite.

...
break;

case SB_PAGEDOWN :

Défilement d'une page vers le bas ou vers la droite.

...
break;

case SB_THUMBPOSITION :

L'utilisateur a sélectionné l'élévateur et l'a positionné à un endroit quelconque sur la Scroll Bar.

...
break;
}

A.6. Les ressources

Les ressources sont un ensemble de moyens dont une application a besoin pour mener à bien sa tâche. Elles sont rassemblées dans un fichier ".RC" et sont compilées grâce au Resources Compiler. Les différents types de ressources dont une application peut avoir besoin sont des icônes, des curseurs, des Bitmaps, des fontes de caractères, des menus, des chaînes de caractères, des accélérateurs et des Dialog Boxes.

Une icône est un Bitmap utilisé par l'application soit quand elle est mise en attente (dans la Icon Area), soit pour effectuer un dessin dans la Client Area, soit pour mettre dans une Dialog Box ou une Message Box.

Un curseur est la représentation de la position de la souris à l'écran. L'application peut changer de curseur pour indiquer par exemple à l'utilisateur le mode de travail dans lequel il se trouve, la fenêtre dans laquelle il se trouve, ...

Un Bitmap a la même utilité qu'une icône dans le cas de son utilisation comme primitive de dessin dans la Client Area. Il peut également être utilisé comme item dans un menu.

Les menus peuvent être définis de deux types : les Menuitems et les popup menus. En fait, les Popup menus sont composés de Menuitems qui se déroulent quand l'utilisateur enfonce le bouton de la souris sur l'identifiant du menu.

Les Dialog Boxes sont des fenêtres qui contiennent une ou plusieurs fenêtres de contrôle ou Control Window. Une fenêtre de contrôle est une Child Window prédéfinie que n'importe quelle application peur utiliser pour effectuer ses entrées/sorties.

Comme pour n'importe quelle fenêtre, une Window Function doit être associée à une Dialog Box.

Illustration de ressources :

Pour chaque type de ressource, nous allons présenter un exemple extrait du fichier des ressources et la manière de l'utiliser dans un programme d'application.

L'icône :

fichier des ressources :

```
des  ICON   des.ico
```

des est le type de l'icône, ICON est le mot clef spécifiant que ce qui suit est une icône et des.ico est le nom du fichier contenant le Bitmap représentant l'icône.

programme :

```
pDesClass->hIcon = LoadIcon(hInstance, (LPSTR)"des");
```

Charge l'icône de type "des" (l'exemple est extrait de la définition d'une classe de fenêtre).

```
Icon = LoadIcon(hInstance, IDI_QUESTION);
```

Charge une icône prédéfinie (dans l'exemple le point d'interrogation).

Le curseur :

fichier des ressources :

```
c_co CURSOR pen.cur
```

c_co est le type du curseur, CURSOR est le mot clef spécifiant que ce qui suit est un curseur et pen.cur est le nom du fichier contenant le Bitmap représentant le curseur.

programme :

```
CurCo = LoadCursor (hInstance, (LPSTR)"c_co");
```

Charge le curseur de type "c_co" (CurCo devient l'identifiant de ce curseur).

```
CurMo = LoadCursor (hInstance, IDC_ARROW);
```

Charge un curseur prédéfini (dans l'exemple la flèche inclinée).

```
SetCursor (CurWnd);
```

CurCo devient le curseur courant.

Le menu :

fichier des ressources :

```
des MENU
BEGIN
    POPUP "Edit"
    BEGIN
        MENUITEM "Cut\tdel"          , IDDCUT    , GRAYED
        MENUITEM "Copy\tF2"          , IDDCOPY   , GRAYED
```

```
        MENUITEM "Paste  \tins"      , IDDPASTE  , GRAYED
        MENUITEM SEPARATOR
        MENUITEM "Undo\tF1"          , IDDUNDO   , GRAYED
        MENUITEM "Redo\tF3"          , IDDREDO   , GRAYED
    END
END
```

des est le type du menu, MENU est le mot clef spécifiant que ce qui suit entre le BEGIN et le END est le contenu de la Menu Area. POPUP et MENUITEM définissent le type de menu, "Cut\tdel" est le nom à placer dans la Menu Area représentant une option du menu ("\t" signifie une tabulation), IDDCUT est l'identifiant de cette option, GRAYED signifie que le menu doit être en gris, c'est-à-dire inactif au moment du chargement et SEPARATOR est un mot clef spécifiant qu'il faut tracer une ligne horizontale dans le menu.

représentation de l'exemple :

programme :

```
pDesClass->lpszMenuName = (LPSTR)"des";
```

Charge le menu de type "des" (ligne de programme extraite de la définition d'une classe de fenêtre).

```
hMenu = GetSystemMenu (hWnd, FALSE);
```

Récupère l'identifiant du menu système.

```
hMenu = GetMenu (hWnd);
```

Récupère l'identifiant du menu application.

```
ChangeMenu (hMenu, 0, NULL, 999, MF_APPEND | MF_SEPARATOR);
```

Modifie un menu : ajout d'un séparateur.

```
ChangeMenu (hMenu, 0, (LPSTR)szAbout, IDSABOUT, MF_APPEND |
    MF_STRING);
```

Modifie un menu : ajout de l'option "About ...".

```
EnableMenuItem (hMenu, IDDCUT, MF_ENABLED);
```

Rend l'option d'un menu initialement inactive disponible.

```
EnableMenuItem (hMenu, IDDCUT, MF_GRAYED);
```

Rend l'option d'un menu initialement disponible inactive.

```
CheckMenuItem (hMenu, ToolCheck, MF_CHECKED);
```

Marque l'option d'un menu.

```
CheckMenuItem (hMenu, ToolCheck, MF_UNCHECKED);
```

Supprime la marque d'une option d'un menu.

Les accélérateurs :

fichier des ressources :

```
des ACCELERATORS
BEGIN
    VK_DELETE    , IDDCUT        ,VIRTKEY
    VK_F2        , IDDCOPY       ,VIRTKEY
    VK_INSERT, IDDPASTE          ,VIRTKEY
    VK_F1        , IDDUNDO       ,VIRTKEY
    VK_F3        , IDDREDO       ,VIRTKEY
    VK_F4        , IDDMOVE       ,VIRTKEY
    VK_F5        , IDDCHANGE     ,VIRTKEY
    VK_F6        , IDDPLACE      ,VIRTKEY
    VK_F7        , IDDCONNECT    ,VIRTKEY
    VK_F8        , IDDSELECT     ,VIRTKEY
END
```

des est le type de la liste d'accélérateurs, ACCELERATOR est le mot clef spécifiant que ce qui suit entre le BEGIN et le END est la liste d'accélérateurs. La première ligne associe à la touche "Delete" la fonction de l'option du menu identifiée par IDDCUT.

programme :

```
hAccelTable = LoadAccelerators(hInstance, (LPSTR)szAppName1);
```

Charge les accélérateurs identifiés par des dans la table hAccelTable.

```
TranslateAccelerator(hWnd, hAccelTable, (LPMSG)&msg) == 0)
```

Si la touche enfoncée est un accélérateur, la traduit comme étant une sélection de l'option du menu correspondante.

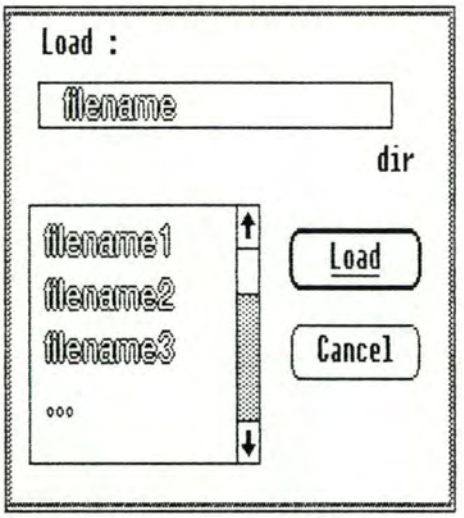
La Dialog Box :

fichier des ressources :

```
LOADBOX DIALOG 40, 30, 135, 120
STYLE WS_POPUP | WS_DLGFRAME
BEGIN
     LTEXT "Load : "    -1, 9, 2, 47, 8
     EDITTEXT           IDTXT, 8, 15, 110, 12, ES_AUTOHSCROLL | ES_LEFT |
                                                WS_BORDER
     RTEXT ""                                  IDDIR, 50, 30, 75, 10
     LISTBOX                                   IDLIST, 5, 44, 72, 70, WS_TABSTOP
     DEFPUSHBUTTON "Load"  IDLOAD, 86, 52, 41, 15, WS_GROUP
     PUSHBUTTON "Cancel"   IDCANCEL, 86, 75, 41, 15, WS_GROUP
END
```

La Dialog Box est identifiée par LOADBOX. Sa coordonnée par rapport au coin supérieur gauche de la Client Area de sa Parent Window est (40, 30) et sa taille est de (135, 120). Elle possède les Control Windows suivantes : un texte justifié à gauche (LTEXT) ayant la valeur "load :", une zone d'édition de texte (EDITTEXT) identifiée par IDTXT, un texte justifié à droite (RTEXT) identifié par IDDIR, une List Box (LISTBOX) identifiée par IDLIST et deux PUSHBUTTON, l'un identifié par IDLOAD (celui par défaut) et l'autre identifié par IDCANCEL. Les quatres valeurs suivant chaque identifiant sont, respectivement, les coordonnées de la Control Window, relatives au coin supérieur gauche de la Dialog Box, sa longueur et sa hauteur.

représentation :



programme :

```
BOOL FAR PASCAL Load (hDlg, message, wParam, lParam)
```

Window Function associée à la Dialog Box "LOADBOX".

```
HWND    hDlg;
unsigned message;
WORD    wParam;
LONG    lParam;
{
      if (message == WM_COMMAND)
```

L'utilisateur a effectué une action dans une Control Window.

```
      {
            switch(wParam)
```

wParam contient l'identifiant de la fenêtre de contrôle dans laquelle l'utilisateur a effectué son action.

```
{
  case IDTXT:
```

Envoyé chaque fois que l'utilisateur frappe un caractère ou modifie la chaîne de caractères affichée (del, ins, ...).

```
      GetDlgItemText (hDlg, IDTXT, (LPSTR)LoadFile, 80);
```

Saisit la chaîne de caractères.

```
      AnsiUpper ((LPSTR)LoadFile);
```

La transforme en majuscules.

```
      if (LoadFile[0] == '\0')
```

Rend le pushbutton "Load" inactif si la chaîne de caractères est vide, le rend actif sinon..

```
            EnableWindow (GetDlgItem (hDlg, IDLOAD), FALSE);
      else
            EnableWindow (GetDlgItem (hDlg, IDLOAD), TRUE);
  break;

  case IDLIST:
```

Envoyé chaque fois que l'utilisateur sélectionne un item dans la List Box.

```
      bDir = DlgDirSelect (hDlg, (LPSTR)NewLoadFile, IDLIST);
```

Saisit l'item sélectionné et positionne bDir à TRUE si l'item est un directory.

```
            if (bDir == TRUE)
            {
                Changement de directory
            }
            strcpy(LoadFile, NewLoadFile);
            SetDlgItemText (hDlg, IDTXT, (LPSTR)LoadFile);

                Répercute l'item choisi dans l' "Edit Box".

        break;

        case IDOK :
        case IDLOAD :

            Envoyé quand l'utilisateur sélectionne le Defpushbutton "Load".
            Effectue le chargement proprement dit. IDOK est l'identifiant
            envoyé par défaut quand l'utilisateur enfonce la touche
            <RETURN>

        break;

        case IDCANCEL :

            Envoyé quand l'utilisateur sélectionne le Pushbutton "Cancel".

            EndDialog (hDlg, TRUE);

                Efface la Dialog Box et rend la main à sa Parent Window.

        break;

        default :
        break;
    }
    return TRUE;
}
```

```
else if (message == WM_INITDIALOG)
```

Message envoyé quand l'utilisateur invoque la Dialog Box (premier message envoyé).

```
{
    strcpy (LoadFile, "*.DBF");
    SetDlgItemText (hDlg, IDTXT, (LPSTR)LoadFile);
```

Initialise l'Edit Box avec "*.DBF".

```
    DlgDirList (hDlg, (LPSTR)LoadFile, IDLIST, IDDIR, (WORD)0X4010);
```

Initialise la LISTBOX : Elle contiendra tous les fichiers du type "*.DBF" ainsi que tous les sous-directories. Le nom du directory courant est placé dans IDDIR.

```
    return TRUE;
}
else return FALSE;
}
```

A.7. Le GDI (Graphics Device Interface)

A.7.1. Les coordonnées

Le GDI est la partie de MS-WINDOWS responsable de toutes les actions graphiques. Le GDI, pour rester indépendant de tout système physique, utilise un type de coordonnées internes : les Logical Coordinates.

Quel que soit le contexte, ces coordonnées logiques s'étendent, aussi bien en X qu'en Y, de -32768 à 32767. L'origine est donc au centre. Les coordonnées physiques du système dépendent du périphérique. Néanmoins, le GDI leur impose de s'étendre en X et en Y et d'avoir leur origine dans le coin supérieur gauche. L'affichage d'une image (à l'écran, sur imprimante, ...) consiste donc en une transformation de coordonnées logiques en coordonnées physiques.

A.7.2. Le Display Context

Le Display Context est un ensemble de données décrivant un périphérique graphique. Il est continuellement mis à jour par le GDI. Un cas particulier est le Display Context de la Client Area d'une fenêtre . Il doit être connu pour effectuer n'importe quelle opération graphique. Il peut être aisément récupéré par la fonction

```
hDC = GethDC (hWnd);
```

où hDC est un pointeur vers la structure contenant les informations, et hWnd est l'identifiant de la fenêtre. Une fois les dessins terminés, le Display Context doit être libéré par la fonction

```
ReleaseDC (hWnd, hDC);
```

Le Display Context comprend un ensemble d'attributs qui peuvent être modifiés par l'utilisateur. Le lecteur trouvera à la page 141 du manuel de référence l'ensemble des valeurs par défaut. Nous allons illustrer comment modifier certains de ces paramètres.

```
hpen = GetStockObject (WHITE_PEN);
```

GetStockObject retrouve l'identifiant d'un objet prédéfini. Cet objet peut être du type PEN (pour tracer une ligne par exemple), BRUSH (pour colorier une région) ou FONT (caractères). Dans ce cas, il s'agit d'une plume de couleur blanche.

```
hpenOld = (HPEN) SelectObject (hDC, (HANDLE) hpen);
```

SelectObject permet de remplacer l'objet courant par un autre. La fonction retourne un pointeur vers l'ancien objet.

```
SelectObject (hDC, GetStockObject (NULL_BRUSH));
```

Cette fonction correspond à la combinaison des deux fonctions précédentes dans le cas d'une BRUSH.

```
OldDrawMode = SetROP2 (hDC, R2_XORPEN);
```

SetROP2 permet de modifier le mode de dessin courant. Le GDI utilise ce mode de dessin pour combiner les plumes et l'intérieur des objets qui auraient été coloriés. R2_XORPEN signifie qu'un trait doit être dessiné dans la couleur de la plume s'il n'y a rien sur le Background, dans la couleur opposée sinon.

```
hFont = CreateFont (8, 8, 0, 0, 400, (char)NULL, (char)NULL, (char)NULL,
                    (short)ANSI_CHARSET, (short)OUT_DEFAULT_PRECIS,
                    (short)CLIP_DEFAULT_PRECIS, (char)DEFAULT_QUALITY,
                    (char)VARIABLE_PITCH | FF_DECORATIVE, (LPSTR)NULL);
```

La fonction CreateFont permet de créer un type de caractère logique ayant, dans ce cas, les caractéristiques suivantes.

8 : hauteur du caractère

8 : largeur moyenne du caractère

0 : angle en dixième de degré de chaque de texte par rapport au bas de la page

0 : angle en dixième de degré entre la ligne de base du caractère et l'axe des X

400 : l'épaisseur du caractère (entre 0 et 1000) : 400 = normal, 700 = gras

NULL : le caractère n'est pas en italique

NULL : le caractère n'est pas souligné

NULL : le caractère n'est pas barré

ANSI_CHARSET : utilisation de jeux de caractères respectant le standard ANSI

OUT_DEFAULT_PRECIS : précision de l'affichage

CLIP_DEFAULT_PRECIS : précision de la coupure des caractères quand on sort de la région spécifiée

DEFAULT_QUALITY : qualité de l'affichage

VARIABLE_PITCH | FF_DECORATIVE : spécifie l'espacement et la famille du caractère

NULL : chaîne de caractère spécifiant le nom du caractère à utiliser

```
oldfont = SelectObject (hDC, hFont);
```

Remplace le Font par défaut par le Font qui vient d'être créé.

```
SetBkMode (hDC, TRANSPARENT);
```

SetBkMode sélectionne le mode selon lequel le Background doit réagir. TRANSPARENT signifie que le fond doit rester inchangé lors du dessin de

lignes, de hachures ou de texte, OPAQUE signifie qu'il doit être redessiné dans sa couleur.

```
SetBkColor (hDC, (LONG) 0XFFFFFF);
```

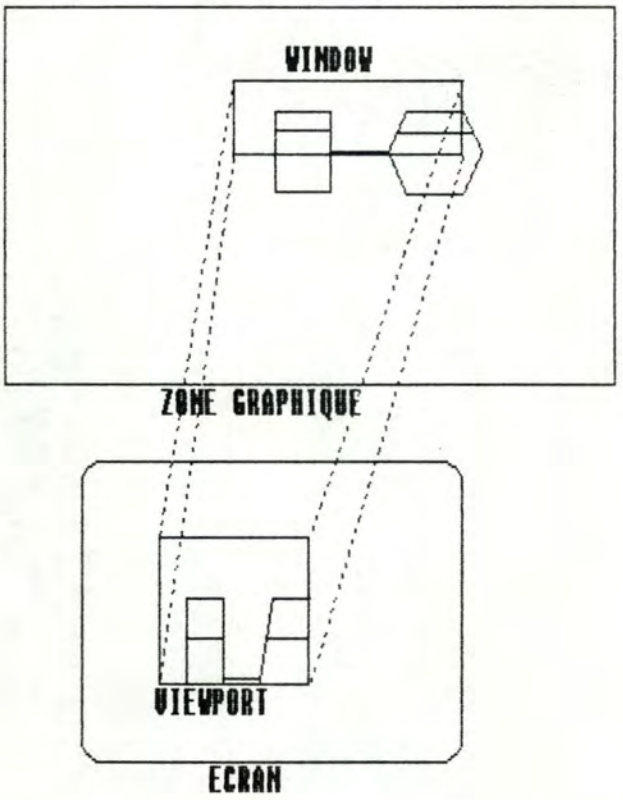
Cette fonction défini la couleur du Background.

```
SetMapMode (hDC, MM_ANISOTROPIC);
```

SetMapMode défini comment les coordonnées logiques sont transformées en coordonnées physiques. MM_TEXT (valeur par défaut) signifie qu'une unité logique correspond à un pixel. MM_ANISOTROPIC signifie que les unités logiques sont transformées arbitrairement, indépendamment en X et en Y, en fonction de la taille et de la fenêtre du Viewport.

```
SetWindowOrg (hDC, SizeWnd.left + whpos, SizeWnd.top + wvpos);
SetWindowExt (hDC, SizeWnd.right, SizeWnd.bottom);
SetViewportOrg (hDC, SizeWnd.left, SizeWnd.top);
SetViewportExt (hDC, SizeWnd.right, SizeWnd.bottom);
```

Ces quatres fonctions permettent de définir l'origine et la taille respectivement de la fenêtre et du Viewport.



A.7.3. Les fonctions d'affichage

Moyennant l'utilisation du Display Context, MS-WINDOWS offre à l'utilisateur un ensemble de fonctions permettant de créer des images sur une surface d'affichage. Les fonctions que nous avons utilisées nous ont toutes semblées très simples d'utilisation. Nous nous contenterons de présenter quelques exemples.

```
MoveTo (hDC, 100, 100);
```

Cette fonction positionne le curseur aux coordonnées logiques (100, 100)

```
LineTo (hDC, 200, 100);
```

Cette fonction permet de tracer une ligne de la position courante jusqu'aux coordonnées logiques (200, 100)

Rectangle (hDC, rect.left, rect.top, rect.right, rect.bottom);

Cette fonction dessine un rectangle ayant comme coin supérieur gauche les coordonnées logiques (rect.left, rect.top) et comme coin inférieur droit (rect.right, rect.bottom). Le rectangle est colorié en utilisant la BRUSH courante et le bord est dessiné avec la plume courante.

BitBlt (hDC, 30, 30, 36, 36, hDC, x, y, rop);

Cette fonction déplace un Bitmap d'une source (le deuxième paramètre hDC) vers une destination (le premier paramètre hDC). (30, 30) est la coordonnée logique du point supérieur gauche du rectangle dans lequel le Bitmap devra être dessiné, (36, 36) est la largeur et la hauteur de ce rectangle, (x, y) sont les coordonnées de la source et (rop) définit comment le GDI doit combiner les bits du Bitmap source avec ceux se trouvant dans le rectangle destination. L'ensemble des modes possibles sont repris aux pages 411 à 418 du le manuel de référence.